REPORT DOCUMENTATION PAGE

AD-A279 965

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		FINAL/15 JUL 93 TO 14 JAN 94

4. TITLE AND SUBTITLE

(SBIR93-01) A NEURAL EXPERT APPROACH TO SELF SELF DESIGNING FLIGHT CONTROL SYSTEMS

5. FUNDING NUMBERS

3005/SS
F49620-93-C-0050

6. AUTHOR(S)

DR. CAGLAYAN

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

CHARLES RIVER ANALYTICS INC
55 WHEELER ST
CAMBRIDGE MA 02138-1125

8. PERFORMING ORGANIZATION REPORT NUMBER

AFOSR-TR- 94 0310

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

F49620-93-C-0050

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

13. ABSTRACT (Maximum 200 words)

Based on the simulations performed in this phase I study, we show that Hopfield and RBF feedfoward network architectures may have a great potential in the control of nonlinear systems. In particular, Hopfield implementation of Lagrange multiplier method is suitable for real-time adaptive optimal control. Similarly, RBF feedfoward neural network architectures are suitable for learning inverse dynamics and inverse trim in aircraft FCS applications. In addition, RBF feedfoward are easier to train than backpropagation sigmoid networks since RBF formulation results in linear parameters.

The initial simulations we performed show very promising results as exemplified by the small control errors in closed-loop simulations using the nonlinear /A-18 longtitudinal dynamics. Further studies are needed to test the applicability of the techniques to real world problems and to study the robustness, stability and general reliability of the proposed neural techniques. Neural networks by themselves cannot be the panacea to all the nonlinear control problems. An effort has to be made to incorporate all the available knowledge about the dynamics system to achieve good performance.

14. SUBJECT TERMS			15. NUMBER OF PAGES
DTIC QUALITY INSPECTED 5			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

Charles River Analytics

Final Report No. R93081
Contract No. F49620-93-C-0050DEF

A Neural Expert Approach to Self Designing Flight Control Systems

Sherif M. Botros, Alper K. Caglayan and Greg L. Zacharias
Charles River Analytics
55 Wheeler Street
Cambridge, MA 02138

21 April 1994

Prepared for:

Sonia Hudson
Department of the Air Force
Bolling AFB, DC

Charles River Analytics

Final Report No. R93081
Contract No. F49620-93-C-0050DEF

A Neural Expert Approach to Self Designing Flight Control Systems

Sherif M. Botros, Alper K. Caglayan and Greg L. Zacharias
Charles River Analytics
55 Wheeler Street
Cambridge, MA 02138

21 April 1994

Prepared for:

Sonia Hudson
Department of the Air Force
Bolling AFB, DC

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

As modern high-performance aircraft improve in performance and maneuverability, their design becomes more and more statically unstable. These aircraft depend on inner-loop stability/control system augmentation to increase bare airframe stability while providing the pilot with high performance and superior maneuverability under a wide range of flight conditions. This poses a challenge to the flight control system (FCS) designer: the extreme range of flight conditions introduces significant uncertainties and nonlinearities that the FCS design must allow for. In addition, the FCS must be adaptive and reconfigurable, in order to maintain stability if one of the control effectors fails.

The current FCS design approach is 1) to generate linearized models of the flight dynamics for a large set of trim conditions, 2) next to use linear control-system theory to design controllers and controller gains that are valid only for a limited region of the state/control space around the trim point, and 3) to design a gain schedule by interpolating the gains between the different trim conditions. While such a procedure has resulted in satisfactory FCS performance in the past, it has many disadvantages: a time-consuming and expensive trial-and-error process; a lack of adaptability of the design to changes in the dynamics; a difficulty in handling extremely nonlinear flight conditions, such as what occurs at high angles of attack; and a tendency toward a conservative design that improves robustness to uncertainties at the expense of reduced maneuverability.

One alternative to linear control methods is to use neural networks to control nonlinear systems. An artificial neural network (ANN) approach to FCS design might provide a means of eliminating or reducing many of the disadvantages of control methods that rely on linearized models. Recently proposed types of neural networks can accomplish complex tasks-such as pattern classification, function approximation and generalization from examples, content-addressable information retrieval, error correction, optimization, adaptation, and learning. These abilities of neural networks might provide several potential advantages over the conventional control-design methods of FCS systems:

- Neural networks can approximate nonlinear smooth mappings arbitrarily closely, and this might provide accurate models and nonlinear controllers that can achieve superior performance and maneuverability over a wide range of flight regimes.
- Neural networks offer not only a rapidly adaptable but also an on-line learning solution. Hence, neural networks offer benefits beyond those of adaptive control techniques. Rapid adaptation capability has a significant value in a high-performance operational environment with aircraft configurations, stores and missions that change constantly, while the learning

capability is crucial for robustness under hardware failures and control surface/body damage.

- Optimizing neural networks, such as Hopfield-like recurrent networks, might give real-time adaptive solutions to many of the control design problems. Many of these problems (e.g., pole assignment, optimal control design, parameter estimation and state observer design) can be formulated as optimization of suitable objective functions. Optimizing neural networks can provide a cheap real-time on-board solution to these optimization problems. This will reduce the need for predesigned control and increase adaptability to both changes in mission requirements and changes in dynamics, such as those that might result from battle damage.
- Neural networks can improve productivity in the off-line design of FCSs. ANN's can automate every stage of the design process that involves trial-and-error. For example, an ANN might be trained to identify the quality of a particular design, based on some parameters. This network can then find the controller parameters that yield a good FCS design.
- ANNs can improve implementation efficiency on the emerging neural computers. Given the availability of relatively inexpensive VLSI designs for neural network structures, FCS designs which could not be implemented in real-time on serial machines can now be considered for practical applications.

1.1 Applications of Neural Networks in Control

There are many different ways that neural network techniques might help solve control problems. Some of these techniques are summarized by Atkeson (1991). A recent survey for the application of neural networks in control is given by Hunt, Sbarbaro, Zbikowski, et al., Hunt, Sbarbaro, Zbikowski, et al. (1992). Over the past few years, there have been many attempts at applying neural networks in control, using different control and network architectures, with varying degrees of success. Among the recent papers that have examined the application of neural networks in control are Psaltis, Sideris and Yamamura, Psaltis, Sideris and Yamamura (1988); Marzuki and Omatu, Marzuki and Omatu (1992); Narendra and Mukhopadhyay (1992); Schiffman and Geffers, Schiffman and Geffers (1993); Chen and Khalil, Chen and Khalil (1992); Pao, Phillips and Sobajic, Pao, Phillips and Sobajic (1992); Levin and Narendra (1993); Kuschewski, Hui and Zak, Kuschewski, Hui and Zak (1993); Sanner and Slotine (1992); Jordan and Rumelhart (1992); Atkeson, (1991), and the papers cited therein. We will summarize in this section some of the proposed neural-network applications in control.

An inverse model is the most direct way for using a neural network as a controller. The inverse neural network model is simply an associative network, having as its input the desired output(s) x_d and the current states of the system x, and as its output the control action(s) that should be applied. An inverse model of the dynamics can be used in a variety of configurations in a control system. These different configurations will be discussed in a later chapter.

Another possible application for neural networks in control is to encode the outcome of a control action as a function of the states and controls. This has been called forward modeling. Unlike inverse models, forward models, by definition, always exist for deterministic systems. In the case of forward models, finding the control command to use in order to reach a certain desired state from a given state is not as straightforward as it is in the case of the inverse model. Root finding or optimization techniques may be used to solve for the control commands as a function of the states and desired change in the states Atkeson (1991). Hoskins, Hwang and Vagners, Hoskins, Hwang and Vagners (1992) use back propagation techniques for the iterative inversion of the forward model, and they propose its application in adaptive control. Forward models serve as predictors of the behavior of the dynamic systems they model and can optimize controllers.

There might be more than one neural network in a single control system. For example, forward models can be used for a predictor model and an inverse model for a controller. The role of the plant-dynamics forward model is to predict the response of the system and to propagate back the error in the output (Jordan, 1989). More generally, it can propagate back any performance gradient in order to adjust the controller parameters. An inverse model might make an initial guess for the root finding algorithm for the forward model and correct the errors in approximating the forward model.

Neural networks used as pattern classifiers might also have applications in control. The classifier acts as a nonlinear switch between a discrete set of controllers that is based on measurements of the states and desired outputs of the system and the desired mission. The switch can also change smoothly from one controller to the next. This can be thought of as a more generalized method of gain scheduling. The controllers themselves need not be fixed and can be some other function approximators or neural networks. This type of use of neural network classifiers in control may be useful, for example, when the controller function varies considerably in the different areas of the state space or depends heavily on the desired mission. This is one possible application for the competitive networks paradigm described by Jacobs et al. (Jacobs, Jordan, Nowlan, et al. 1991; Jacobs and Jordan 1993). Narendra and Mukhopadhyay, (1992) have also suggested the use of neural network classifiers as a switch to select a controller for the case when it is known a priori that the controlled plant can only be in one of a finite number of configurations.

Neural networks can be state observers for state feedback control when not all the states can be measured. Recurrent neural networks have been previously proposed and tested for such tasks. The states encoded with such networks may not necessarily have a physical meaning, but can be a nonlinear function of meaningful state variables.

Another potential use of neural networks in control is to build a model of some measure of performance of the system as a function of some parameters and then use this model to find the parameters that optimize this measure of performance, using nonlinear optimization techniques.

Recurrent neural networks of the Hopfield type might serve as cheap real-time computing elements. Variants of such networks have been proposed for continuous and combinatorial optimization and linear algebra problems such as computation of matrix inverses, the solution of general matrix equations, the estimation of eigenvalues and singular value decomposition (Cichocki and Unbehauen, 1992).

1.2 Application of ANN in FCS

Specific applications of ANNs in FCS include:

- automatic trim computation
- gain scheduling
- adaptive and optimal control
- identification of nonlinear dynamics
- on-line optimization of handling quality
- self-repairing flight control
- automatic trajectory guidance
- integrated fire/flight control

Some of the potential applications of neural networks and fuzzy logic in flight control are summarized in Steinberg (1992). Recently, there have been many reports describing applications of neural networks to flight control problems. For instance, DiGirolamo, DiGirolamo (1992) trained a feedforward neural network to generate control gain schedules based on measurements of the states. The method was successfully applied to the tracking of the pitch rate of a nonlinear longitudinal F/A-18 model. Similarly, Sadeghi, Tascillo, Simons, et al, (1992) trained a feedforward network as a nonlinear feedback controller. They found that the performance of the neural network was inferior to self-tuning adaptive control law. Linse (1990) modeled the

longitudinal trim state of a commercial transport aircraft . Ahmed-Zaid and colleagues (Ahmed-Zaid, Ioannou, Polycarpou, et al., 1992) modeled the pitch dynamics of an F-16 aircraft using a radial basis functions (RBF) neural network. They then used the neural network model and its partial derivatives to control the pitch dynamics. The neural network approach was found to be superior than a simple linear controller. Caglayan and Allen (1990) trained a multilayer perceptron to model optimum guidance trajectories for the aeroassisted orbital plane change scenario. Calise, Kim, Kam, et al. (1992) trained two neural networks to model the inverse dynamics of the rolling rate. The neural networks were able to generate the differential deflection of the tail surfaces as a function of the roll rate command, the aircraft state vector and the rudder deflection. Rokhsaz and Steck (1993) used feedforward neural networks to model the nonlinear aerodynamics and flight dynamics of aircrafts under different conditions. They conclude that neural networks have good generalization properties for the aerodynamic modeling problems, but do not perform as well in modeling the flight dynamics. Troudet, Garg and Merrill (1993) propose the use of a feedforward neural network as a controller for command tracking and apply it to the control of a linearized model of the longitudinal dynamics of an aircraft. They conclude that although the nominal performance of the neurocontroller is better than a standard H_∞ controller, the stability characteristics are poor. The use of a Hopfield neural network in synthesizing the optimal inputs for a command tracker FCS has been demonstrated in Mears, Smith, Chandler, et al. (1993). Application of neural networks to failure detection problems in flight control has been reported by Caglayan and Allen (1990) and Barron, Celluci and Jordan (1990).

Few of the published reports show negative results, and most of the above reported results show promising use of neural networks for different functions of FCS. We believe that a successful FCS design should exploit the abilities of neural networks, namely the ability to map smooth nonlinear functions with high accuracy given few observations, fast computation, and learning and adaptability, while at the same time incorporating domain specific knowledge about the particular dynamics and the existing FCS design expertise that have accumulated over the years.

1.3 Goals of the Current Study

The objective of the current study is to explore and evaluate the feasibility of using different neural network architectures for a self-designing FCS, one which can continuously optimize performance and accommodate changing mission requirements and failures in hardware and battle damages. We will focus in particular on the neural implementation of three major functions associated with FCS design, namely the modeling of the inverse dynamics and inverse trim, parameter estimation and optimal control. For each of these functions we will develop a neural network implementation, and we will evaluate its performance using a high performance aircraft

simulation. We plan to explore and analyze some of the practical problems associated with the implementation of each of these neural network modules for FCS, and to propose possible alternative solutions.

1.4 A Neural FCS for Longitudinal Dynamics

The proposed self-designing FCS architecture that we will explore is as shown in figure 1.4-1.

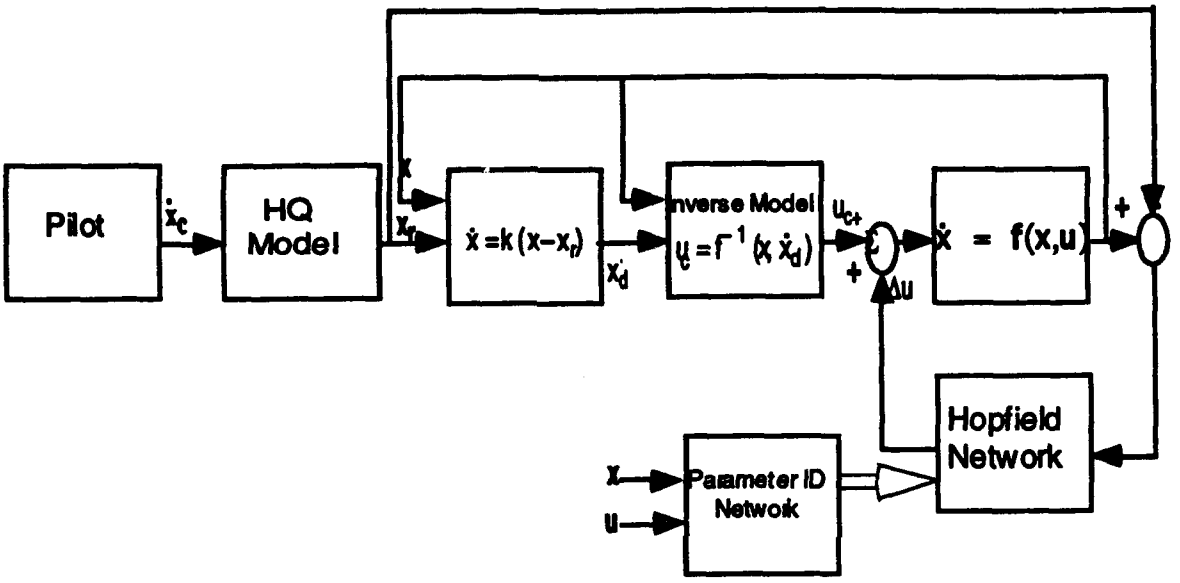


Figure 1.4-1: Proposed Neural FCS Controller Architecture

The inverse model computes the control commands as a function of the desired state trajectory, generated by the pilot and the handling qualities model, and of the measured current state vector. The optimal control network computes an optimal trajectory perturbation based on measurements of the current state vector of the aircraft, an objective function, and the estimated parameters supplied by the parameter estimation module. The parameter ID network provides an estimate of the jacobian of the dynamic response of the system with respect to the state and control vectors, using as input the previous histories of the control and state trajectories.

It is important to emphasize here that the proposed FCS design, shown in figure 1.4-1, is chosen only to show the power of neural networks in FCS design. No attempt is made to optimize the design with respect to the issues of robustness to noise or sensitivity to modeling inaccuracies. Other feedback-control loops might improve stability and robustness in the current neural controller. For instance, the inverse model can be used to linearize the aircraft nonlinear dynamics. A robust optimal linear feedback controller is then used for noise rejection and increased stability of the resultant linear system.

A nonlinear model of the longitudinal dynamics of an F/A-18 aircraft is used to test the developed neural network controllers. The state vector used in the simulations reported here is composed of five state variables: the angle of attack (α), the pitch angle (θ), the pitch rate (q) the total speed (v_t) and the altitude (h). Two control effectors are used in the simulation, the engine thrust (δ_{thr}) and the average elevator deflection (δ_e).

1.5 Achieved Objectives

- We propose and develop different optimizing recurrent neural network architectures which satisfy the dynamic constraints for the optimal control module. We analyze and compare the performance of the different architectures with respect to ease of implementation, accuracy, robustness and speed of computation. We discuss how to implement *limits* on the states and/or controls and how to extend these neural network architectures to find the optimal control of *nonlinear* systems.

- We propose and develop a parameter estimation neural network which can implement optimization-based parameter estimation algorithms. We discuss methods for implementing *forgetting factors* and relationships between parameters. We also propose a feedforward neural network for estimating the jacobian of the nonlinear dynamic equations (stability and control derivatives) based on measurement of the current state and control vectors of the aircraft. We propose a parameter ID module which uses the optimization-based neural network in parallel with a feedforward neural network to reduce the need for dithering signals for parameter identification and help identify failures. We also discuss efficient Hopfield-like neural network architectures for implementing state space parameter identification techniques, that are based on subspace methods (Moonen, DeMoor, Vandenberghe, et al. 1992, Swindelhurst, Roy, Ottersten, et al, 1992).

- We implement and test a radial basis functions neural network (RBF) which models the inverse flight dynamics. The network is found to perform well on training and test sets as well as in tracking simulated trajectories. In addition, we train an RBF neural network to estimate the trim controls and angle of attack, given desired aircraft altitude and speed. We discuss methods of training such a network on-line and incorporating the inverse dynamics in the control loop.

2. THE OPTIMAL CONTROL MODULE

The role of the optimal control module is to find the control inputs which achieve a desired goal optimally. The desired goal is formulated as an objective functional to minimize. For the FCS design proposed in this report and shown in figure 1.3-1, the goal of the optimal controller is only to optimize the control command perturbations (δu) around the current operating point. The output of the inverse model network determines the operating-point control command (u_c) itself. Since the state and control perturbations (δx and δu) around the operating point are assumed to be small, linearized models can be used. The parameter identification module, described in the next chapter, will provide the linearized system parameters that will be used by the optimal controller.

Current linear optimal controller design involves finding the optimal feedback steady-state gains by solving an algebraic Riccati equation, which is a function of the system parameters and the objective functional weights. It must be noted here that the derived feedback gains are only optimal at steady state. For objective functions that have a small time horizon compared to the system dynamics, or are explicit functions of time, the optimal feedback gains are, in general, time varying. For nonlinear systems, many constant linear feedback gains are derived at different operating points, then gain scheduling is used to interpolate the value of the feedback gains to be used at the current operating point.

In the current work, we study a different approach for implementing the optimal controller. Instead of providing the closed-loop optimal feedback gains, the optimal controller module computes the *optimal open-loop* trajectory, based on the current state of the system and the current objective function. The optimal control trajectory is recomputed at each instant of time. Since the optimal trajectory generated always depends on the current state vector, this approach, in essence, is *equivalent to a closed-loop optimal controller*. However, this approach differs from a closed-loop controller in that the equivalent feedback gains here are, in general ,time varying in the case of a limited horizon objective function, even when the system dynamics are linear and time invariant.

In order to achieve closed-loop performance, the open-loop controller should be able to observe the current state of the system and compute the optimal trajectory in real-time. Practically, this is a difficult task due to the amount of computation involved in finding the optimal trajectory. Recently, few researchers have proposed using optimizing neural networks, such as Hopfield networks, to find the optimal trajectory for optimal control problems of linear systems (Lan and Chand, 1990; Mears, et al., 1993). In the current study, we extend the work of previous researchers and discuss different novel approaches for mapping optimal control problems into Hopfield-like neural networks. We explore the advantages and disadvantages of the different techniques. We test the proposed techniques on a linearized F/A-18 dynamics, and we compare the

performance of the Hopfield controllers with the performance of an optimal controller derived by solving the necessary conditions of optimality.

2.1 Function Optimization Using Hopfield Networks

Hopfield networks with continuous valued units belong to the class of recurrent neural networks. A major difference between this class of networks and feedforward neural networks is the presence of dynamics. The architecture of a Hopfield network is shown schematically in figure 2.1-1.

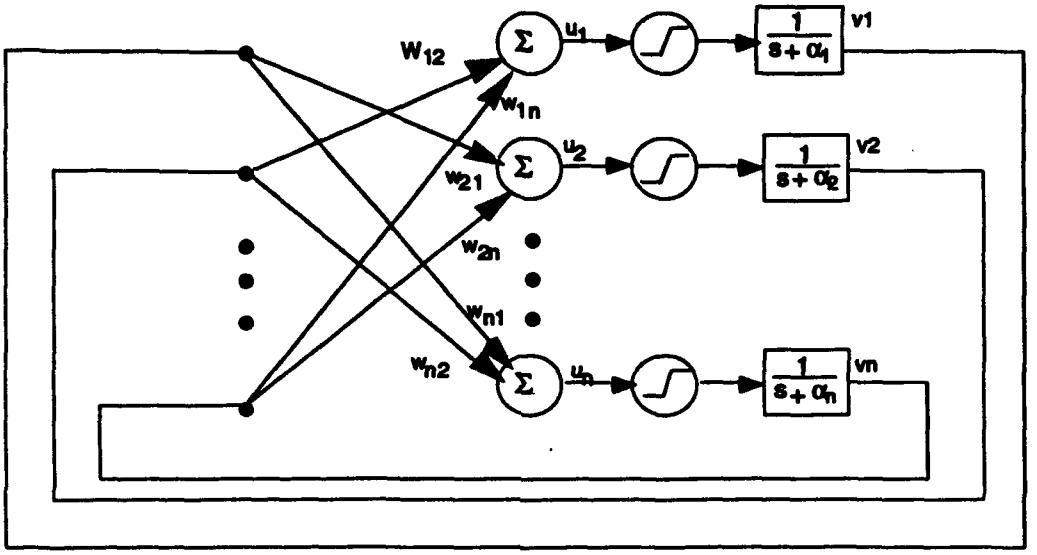


Figure 2.1-1: A Schematic Representation of a Hopfield network

The input to each unit is a weighted summation of the output of all the other units as represented by equation 2.1-1. The output of each unit is a monotonically increasing function of the input, called the activation function.

$$v_i = g(u_i) = g(\sum_j w_{ij} v_j) \qquad (2.1\text{-}1)$$

where u_i and v_i are the input and output of unit i respectively and $g(\cdot)$ is the activation function. The connection weights w_{ij} form a symmetric connectivity matrix. Three different methods have been suggested for finding the output of the different units at steady state: asynchronous, synchronous and continuous. In the asynchronous update, each unit is updated randomly and independently from all other units. In the synchronous update, the units are updated simultaneously at each clock cycle. In the continuous update, the output of each unit is represented by a first-order nonlinear differential equation of the form of equation 2.

$$\tau_i \frac{dv_i}{dt} = -v_i + g(u_i) \qquad (2.1\text{-}2)$$

In this study, we are only interested in the continuous update, since it is easy to implement it using analog hardware. It has been shown, using Lyapunov stability theorems, that for a symmetric connectivity matrix with zero diagonal elements, and using a monotonically increasing function, the set of equations (2.1-2) represents a stable system.

To use a Hopfield network for the minimization of a multivariable cost function we follow the following steps (Hertz, Krogh and Palmer, 1991):

1. Define an energy function H(v) which is bounded from below. The minimum of the energy function should correspond to the minimum of the cost function to be optimized.
2. Use $v_j = g(u_j)$, where g(.) is a monotonically increasing function.
3. Use the following update equation:

$$\tau \frac{du_i}{dt} = -\frac{dH(v)}{dv_i} \qquad (2.1\text{-}3)$$

4. The connectivity matrix of the network is determined by the function $dH(v)/dv_i$. For example, if the energy function H(v) has a quadratic form:

$$H(v) = v^T W v \qquad (2.1\text{-}4)$$

then the connectivity matrix is equal to W.

The system of differential equations (2.1-3) will converge to a local minimum of the energy function. In the case of a convex energy function, such as is the case with quadratic objective functions, the equations will converge to the global minimum. For complex nonlinear objective functions, it is possible to improve the quality of the solution by using techniques such as mean field annealing.

2.2 Optimal Control as a Static Optimization Problem

We will restrict our analysis here to discrete dynamic systems. Continuous dynamic systems can be approximated with discrete systems. A discrete linear-quadratic optimal control problem described by the quadratic objective function represented by equation (2.2-1)

$$J(x,u) = x^T(N)\, Q_N\, x(N) + \sum_{k=0}^{N-1} x^T(k) Q_k x(k) + u^T(k) R_k u(k) \qquad (2.2\text{-}1)$$

and the dynamic system equations (2.2-2)

$$x(k+1) = A(k)\,x(k) + B(k)\,u(k) \quad k = 0, \ldots, N-1 \qquad (2.2\text{-}2)$$
$$x(0) = x_0$$

where $x \in \Re^n$ and $u \in \Re^m$, can be formulated as a static constrained optimization problem as shown in equation (2.2-3) and (2.2-4).

$$\min_v J(v) = v^T H v \qquad (2.2\text{-}3)$$

subject to :

$$M^T v = c \qquad (2.2\text{-}4)$$

where:

$$v = \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N) \\ u(0) \\ u(1) \\ \vdots \\ u(N-1) \end{bmatrix} \qquad (2.2\text{-}5)$$

is the N (n + m) vector of state and control variables,

$$H = \left[\begin{array}{ccc|ccc} Q_1 & & 0 & & 0 & \\ & \ddots & & & & \\ 0 & & Q_N & & & \\ \hline & & & R_0 & & 0 \\ & 0 & & & \ddots & \\ & & & 0 & & R_{N-1} \end{array}\right] \qquad (2.2\text{-}6)$$

is an N (n + m) x N (n + m) block diagonal matrix,

$$M^T = \left[\begin{array}{ccc|ccc} I & & & -B & & \\ -A & \ddots & 0 & & \ddots & 0 \\ & \ddots & \ddots & & & \\ 0 & -A & I & 0 & & -B \end{array}\right] \qquad (2.2\text{-}7)$$

is an N n x N (n + m) matrix that defines the system dynamics, and c is an N (n + m) vector that depends on the initial and boundary conditions.

2.3 Hopfield Networks for Optimal Control

We will discuss in this section different possible implementations of the optimal controller using Hopfield-like networks. This work represents an extension to the Hopfield optimal controllers proposed previously by Lan and Chand (1990); and Mears, et al. (1993), which is based on the penalty function methods for constrained optimization. Using computer simulations, we will compare the following three implementations :

- The penalty method
- The gradient projection algorithm
- The Lagrange multipliers methods

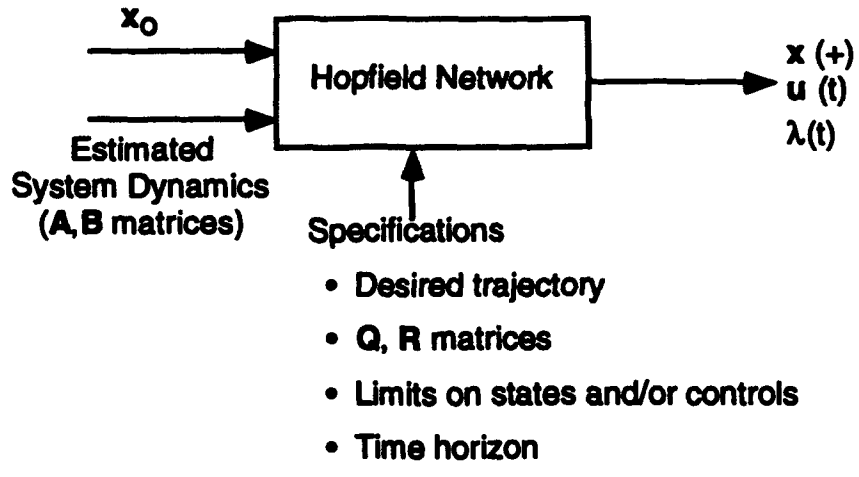


Figure 2.3-1: Optimal Controller

A diagram of a general Hopfield optimal controller is shown in figure 2.3-1. The network uses the estimated system dynamics and the measured current state of the system to compute the optimal trajectories x(t) and u(t) with respect to the given specifications. Depending on the method used, the costate trajectories λ(t) may also be computed as a byproduct of the optimizing network.

2.3.1 Optimal Control using Penalty Functions

Lan and Chand (1990) and Mears, et al. (1993) have proposed transforming the constrained optimization problem (equations 2.2-3 and 2.2-4) into an unconstrained problem, by appending the constraints to the objective function as an additional penalty term, as shown in equation (2.3.1-1) :

$$J'(v) = v^T H v + \kappa \| M^T v - c \|^2 \qquad (2.3.1\text{-}1)$$

where $\kappa >> 0$ is the penalty coefficient. Equation (2.3.1-1) defines an energy function which can then be minimized with respect to v using equation (2.3.1-2)

$$F(dv,dt) = - \in \frac{\partial J'}{\partial v} \qquad (2.3.1\text{-}2)$$

Equation (2.3.1-2) can be implemented using a recurrent Hopfield-like neural network with N(n+m) processing units, representing the components of the vector v. Stability is easy to prove, since equation (2.3.1-2) represents a gradient descent equation to an energy function which has a lower bound. Since the energy function J' is a convex function, convergence to the global minimum of J' is guaranteed. However, it must be kept in mind that the global minimum of J' is only an approximation to the correct global minimum of the objective function J. The connectivity matrix and bias inputs for the different processing units can be derived from the jacobian matrix $\frac{\partial J'}{\partial v}$ by expanding equation (2.3.1-2), as shown below:

$$\frac{dv}{dt} = - W v + M c \qquad (2.3.1\text{-}3)$$

where $W = H + \kappa M M^T$ represents the connectivity matrix and M c represents a bias input.

There are some disadvantages to the above implementation of optimal control. These disadvantages are mainly due to the penalty function implementation of the constraints. In order not to violate the dynamic constraints, the penalty coefficients should be very high. This makes the optimization problem ill-conditioned and consequently results in very slow convergence to the optimal point. Reducing the penalty coefficient on the other hand may result in completely erroneous results, due to the possible violation of the dynamic constraints. A partial solution to this problem is to use a time varying penalty coefficient. The value of the penalty coefficient is gradually increased, as the solution approaches the optimal point.

Although the penalty function technique may work well for constrained optimization problems with few constraints, it does not yield good results when the number of constraints is very large, as is the case of optimal control with a long time horizon.

2.3.1.1 Flight Control Example

To illustrate the problems associated with the penalty method in finding the optimal control trajectories, we tested its performance on an aircraft optimal control problem. The aircraft model used is a 5th order linearized model, obtained by trimming a nonlinear model of the longitudinal dynamics of an F/A-18 aircraft at an altitude of 40,000 ft. and a speed of 700 ft/sec. The states are

the perturbations in speed, angle of attack, pitch angle, pitch rate and altitude. The control variables are the perturbations in the stabilator angle (δe) and thrust (δ_{thr}). The goal is to increase the altitude by 1000 ft. in 5 sec., with minimum perturbations in the other aircraft states. A ramp function with slope of 200 ft / sec is used to represent the change in altitude. The aircraft dynamics were sampled every 0.1 sec. This results in a Hopfield network with 350 units. Two different simulations are shown in figure 2.3.1.1-1 and 2.3.1.1-2 for a penalty coefficient equal to 10^8 and 10^{10} respectively. Shown in the figures are the steady state values achieved by the networks (solid lines) compared to the exact optimal trajectories (dotted lines). As shown in the figures, even with a penalty as high as 10^{10} there is a big discrepancy between the exact optimal trajectories and those achieved by the networks. It is also important to note that at a value of the penalty equal to 10^{10} the network convergence is extremely slow and may not be useful for real-time applications.

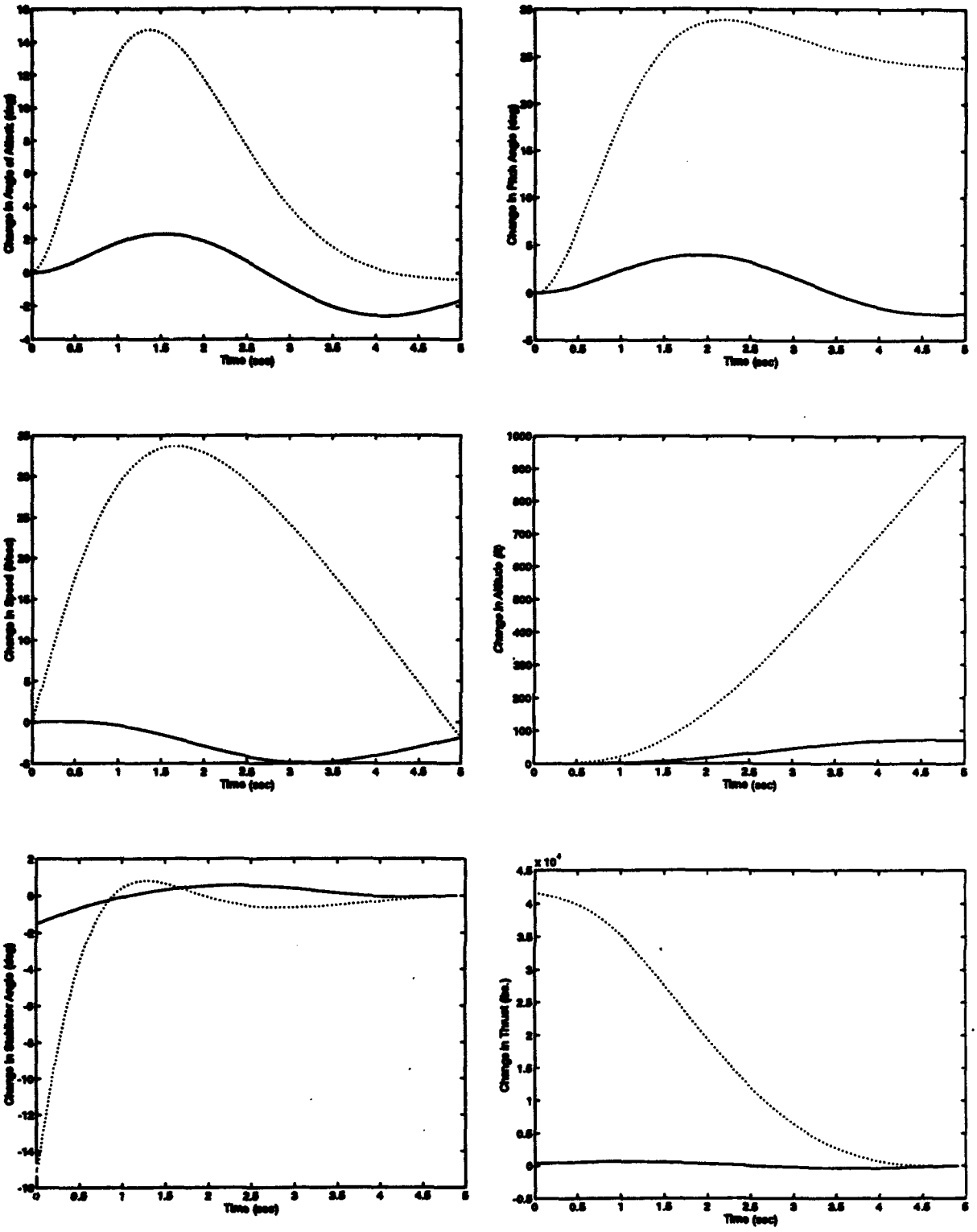


Figure 2.3.1.1-1: Optimal Trajectories Using A Penalty Hopfield Network With K = 10^8

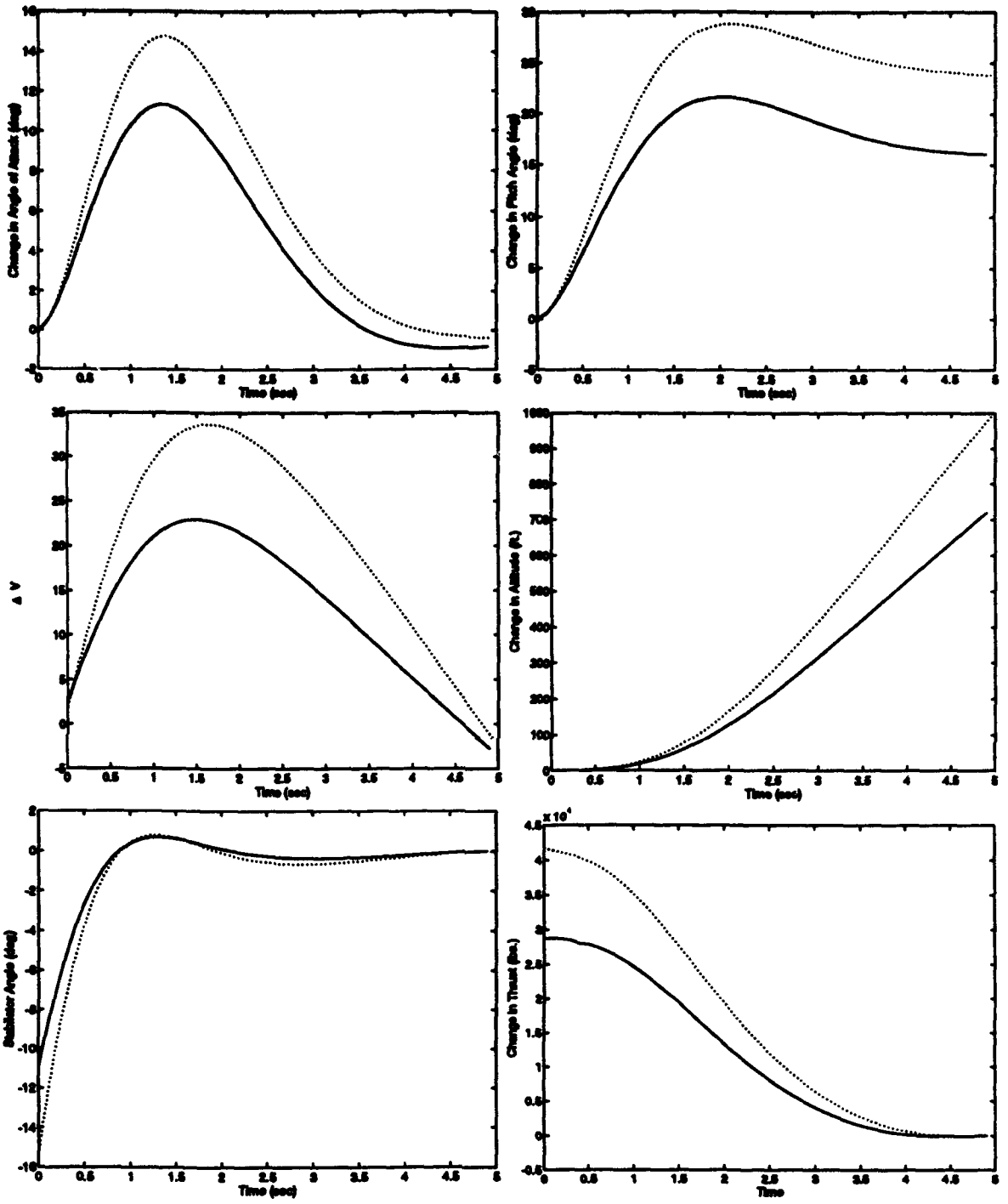


Figure 2.3.1.1-2: Optimal Trajectories Using A Penalty Hopfield Network With K = 10^{10}

2.4 Constraint-Satisfying Hopfield Networks

In this section, we describe a class of Hopfield-type networks which converge to an equilibrium point that satisfies the linear constraints that represent the dynamic system (2.2-4)

exactly. These networks are based on well-known constrained optimization techniques. Convergence to the global optimum and stability can be proven for the case of linear systems with quadratic costs. In this chapter, we will present two networks, one based on the gradient projection algorithm and the second based on the Lagrange multipliers methods.

2.4.1: Gradient Projection Hopfield Network

One possible exact numerical solution to the optimal control problem is through the use of gradient projection (Kirk, 1970). The idea behind this approach is to project the gradient descent equation of the objective function into the hypersurface representing the dynamic constraints, as summarized below.

Define the projection matrix

$$\mathbf{P} = \mathbf{I} - \mathbf{M}\,(\mathbf{M}^T\mathbf{M})^{-1}\,\mathbf{M}^T \quad (2.4.1\text{-}1)$$

The projection matrix projects any change in the vector $\mathbf{v}$ to the hypersurface of the constraints, provided that we start from a valid solution that satisfies the constraints. The projection matrix is symmetric, idempotent and positive semidefinite. If we then use the update rule (2.4.1-2), it is possible to prove that the equations will converge to the globally optimal solution, given an initial feasible trajectory.

$$\frac{d\mathbf{v}}{dt} = -\epsilon\ \mathbf{P}\frac{\partial J}{\partial \mathbf{v}} \quad (2.4.1\text{-}2)$$

which can be written as

$$\frac{d\mathbf{v}}{dt} = -\epsilon\ \mathbf{W}\,\mathbf{v} \quad (2.4.1\text{-}3)$$

where $\mathbf{W} = \mathbf{P}\,\mathbf{H}$ defines the connectivity matrix. The value of the coefficient ϵ and the connectivity matrix $\mathbf{W}$ determine the rate of convergence to the optimal trajectory.

2.4.1.1 Equivalence Between Gradient Projection and the Penalty Method

It is important to note here that the gradient projection method can be viewed as a modified penalty method with an infinite penalty coefficient, hence an exact solution, but at the same time possesses a relatively well-conditioned connectivity matrix. This result can be shown by changing the update rule (2.3.1-3) for the penalty method, to make it well-conditioned without changing the equilibrium point. This can be easily done by multiplying the right hand side of equation (2.3.1-3) by a nonsingular square matrix $\mathbf{F}$ to obtain the following equation :

$$\frac{d\mathbf{v}}{dt} = -\mathbf{F}\,\mathbf{W}\,\mathbf{v} + \mathbf{F}\,\mathbf{M}\,\mathbf{c} \quad (2.4.1.1\text{-}1)$$

Since the matrix **F**, which has the same dimension as **W**, is chosen to be nonsingular, it does not alter the equilibrium points of the original dynamic equations. It is only added to improve the eigenvalue ratio of the linear system of equations (2.3.1-3). Since the ill-conditioning of the connectivity matrix is mainly due to the penalty coefficient, a simple way to reduce this ill-conditioning is by approximately canceling the effect of κ. One possible choice of **F** for performing this cancellation is given by equation 2.4.1.1-2 :

$$\mathbf{F} = [\mathbf{E} + \kappa \mathbf{M}^T \mathbf{M}]^{-1} \qquad (2.4.1.1\text{-}2)$$

where **E** is a symmetric positive definite matrix. The best value of **E** is of course **H**, which will result in all eigenvalues being equal. Another possibility is to use:

$$\mathbf{E} = \mathbf{I} \qquad (2.4.1.1\text{-}3)$$

where **I** is the identity matrix. Using the matrix inversion lemma (Bertsekas, 1982) and taking the limit as $\kappa \rightarrow \infty$, the matrix **F** reduces to:

$$\mathbf{F} = \mathbf{I} - \mathbf{M}(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T \qquad (2.4.1.1\text{-}4)$$

which is exactly the same as the projection matrix **P** defined above.

Although the gradient projection algorithm gives the exact optimal trajectory and has much better convergence properties than the penalty methods, it has some important disadvantages. One disadvantage is that, unlike the penalty method which results in a sparse connectivity matrix **W**, the connectivity matrix that results from the gradient projection algorithm is not sparse. This may represent a problem for parallel implementation. Another disadvantage is that the gradient projection network should always be initialized to a valid solution. Any initial error in the initialization will result in an error in the final solution. In addition, many of the eigenvalues of the projection matrix **W** will be zero, due to the multiplication with the projection matrix **P**. This makes the network less robust to imprecisions in the connectivity matrix values and requires exact implementation of the connectivity matrix. The high precision and the dense connectivity makes the gradient projection network less attractive for hardware implementation.

2.4.1.2 Flight Control Example

We tested the gradient projection network on the same aircraft optimal control problem used to test the penalty method. The gradient projection network size is the same as the penalty network, although much more dense. The optimal trajectories obtained using the gradient projection matrix are shown in figure 2.4.1.2-1 (solid lines). As predicted the network gives an exact solution. The results shown are after 10^{-4} sec of simulation, using a time constant equal to 10^{-6} sec.

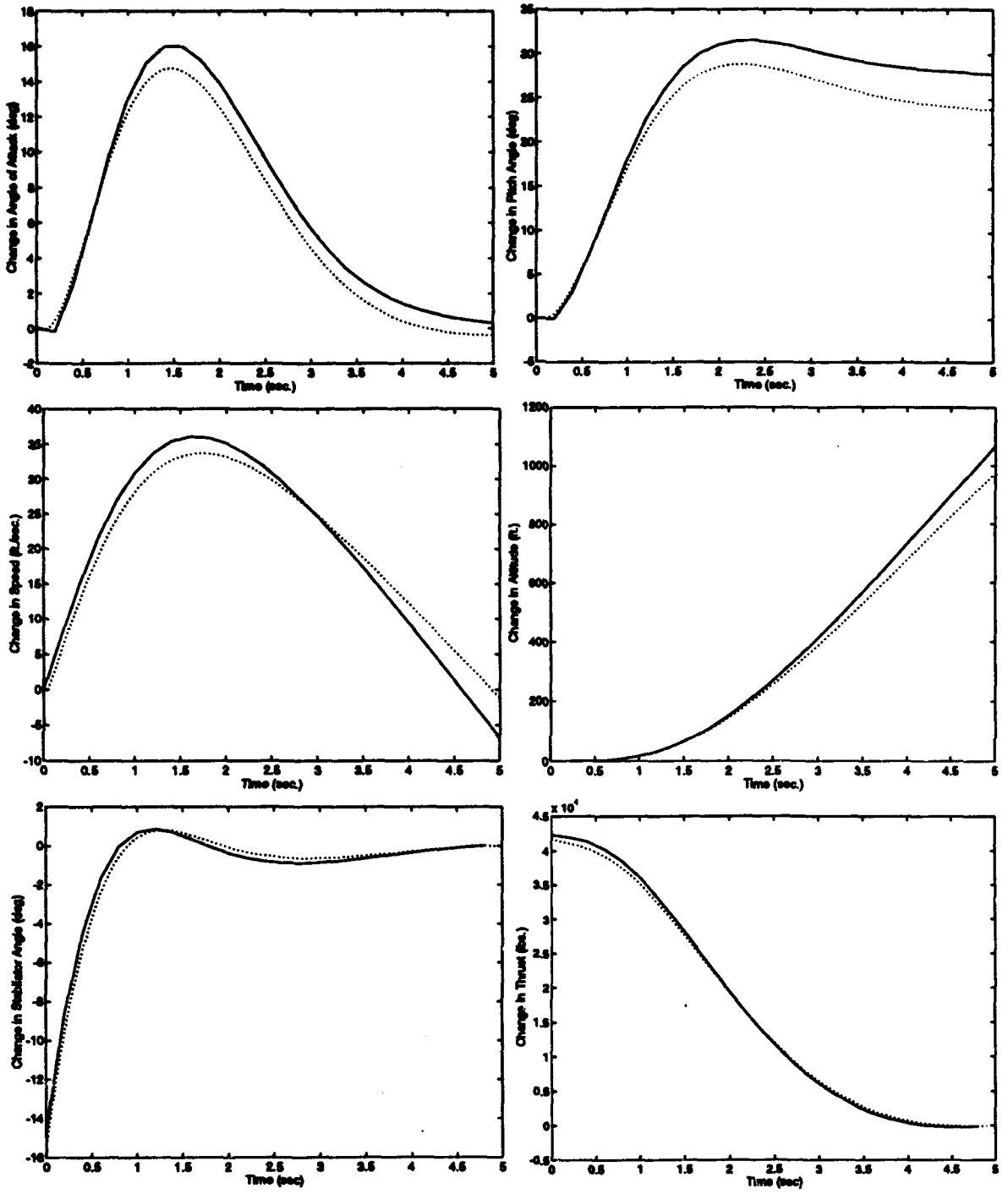


Figure 2.4.1.2-1: Optimal Trajectories using a Gradient Projection Hopfield Network

2.4.2 Lagrange Multipliers Hopfield Networks

Another approach for mapping a discrete linear quadratic optimal control problem to a Hopfield neural network is through the use of Lagrange multipliers.

If we define

$$\mathbf{L} = \begin{bmatrix} \mathbf{H} & -\mathbf{M} \\ -\mathbf{M}^T & \mathbf{0} \end{bmatrix} \tag{2.4.2-1}$$

where the matrices **H**, **M** and **c** have the same definitions as before, then we can express the necessary conditions of optimality (Kirk, 1970) in matrix form as follows:

$$\mathbf{L}\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix} \tag{2.4.2-2}$$

where $\boldsymbol{\lambda}$ is the (N n x 1) vector of the discretized Lagrange multipliers.

The matrix **L** is nonsingular if the optimal trajectory is unique. One possible implementation for the solution of the linear system of equations (2.4.2-2) using a Hopfield network is to construct the energy function

$$\mathbf{E} = \left(\mathbf{L}\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\lambda} \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix}\right)^T \left(\mathbf{L}\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\lambda} \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix}\right) \tag{2.4.2-3}$$

If we then construct the connectivity matrix

$$\mathbf{W} = -\mathbf{L}^T\mathbf{L} \tag{2.4.2-4}$$

and use the update rule :

$$\frac{d}{dt}\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \mathbf{W}\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\lambda} \end{bmatrix} + \mathbf{L}^T\begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix} \tag{2.4.2-5}$$

the dynamic system defined by equation (2.4.2-5) will be stable and will converge to the optimal state, control and costate trajectories.

One serious problem with the above approach is that, although all the eigenvalues of the matrix **W** are guaranteed to be negative and real for positive definite **Q** and **R** matrices, and always stable for a detectable system,* the condition number of the matrix **W** is the square of the condition number of the matrix **L**. This may result in ill-conditioning of the matrix **W** and consequently a very slow convergence to the optimum point.

A better alternative that works well for linear systems with quadratic costs is to use the duality property of the Lagrange function (Bertsekas, 1982), which states that the solution of the constrained minimization problem is equivalent to the minimization over **v** and the maximization

---

* A $(A, \sqrt{Q})$ system is detectable if the unobservable eigenvectors are stable

with respect to λ of the Lagrangian function. We can then use a gradient descent (ascent) algorithm to minimize (maximize) the Lagrangian function with respect to v (λ). This is guaranteed to converge for linear systems with quadratic costs, since the Lagrange function is convex with respect to v. For example, we can choose the connectivity matrix W to be

$$W = \begin{bmatrix} -H & M \\ -M^T & 0 \end{bmatrix} \qquad (2.4.2\text{-}6)$$

In this case the condition number of W will be similar to that of the matrix L. The eigenvalues of W will have a negative real part but they are complex in general, since the matrix W is no longer symmetric. A more complex heuristic update rule for the Lagrange multipliers has been proposed by Barhen, Gulati and Zak (1989). The update rule they propose is harder to implement in hardware, and this study will not pursue it further.

Although the Hopfield network implementation of the Lagrange multipliers method has a bigger size than the gradient projection network, it is much more sparse, which makes it much easier to implement in hardware. In addition, since the connectivity matrix W is derived directly from the system matrices, the computation time required to build the Hopfield network implementing the Lagrange method is very small, which is better for real-time adaptive adjustment of the network weights. A representation of the different inputs and connection strengths associated with each unit is shown schematically in figure 2.4.2-1. In this figure, the circles represent a set of units, for example the state, control and costate vectors at time step k.

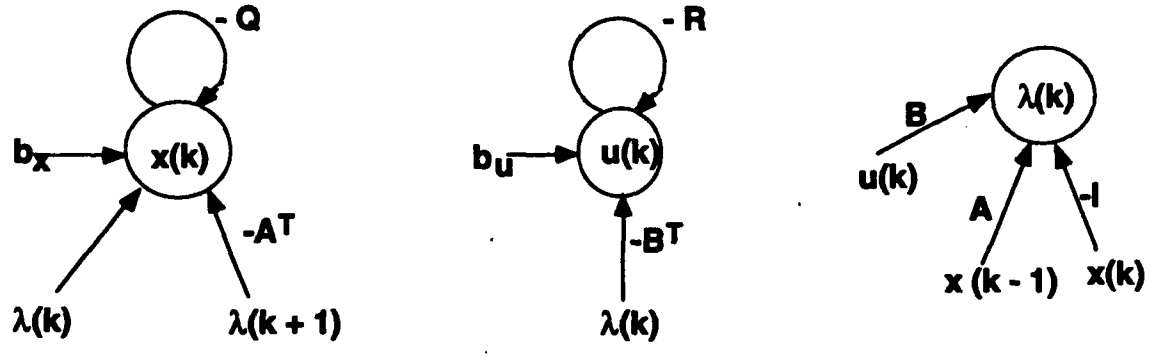


Figure 2.4.2-1: Connectivity of a Lagrange Multipliers Recurrent Network

2.4.2.1 Flight Control Example

We tested the Lagrange multipliers network on the same aircraft optimal control problem used in the previous sections. The network size in this case is larger than the previous networks since the Lagrange multipliers network contain extra units for the costates (Lagrange multipliers). The total number of units for a 0.1 sec. sampling rate equals to 650 units. However the

connectivity of the network is very sparse as compared to the gradient projection network. The optimal trajectories obtained using the Lagrange multipliers network are shown in figure 2.4.2.1-1 (solid lines) together with the exact trajectory obtained using the exact solution of the necessary conditions of optimality. As predicted, the network gives *an almost exact solution*. The results shown are after 10^{-3} sec of simulation, using a time constant equal to 10^{-6} sec.

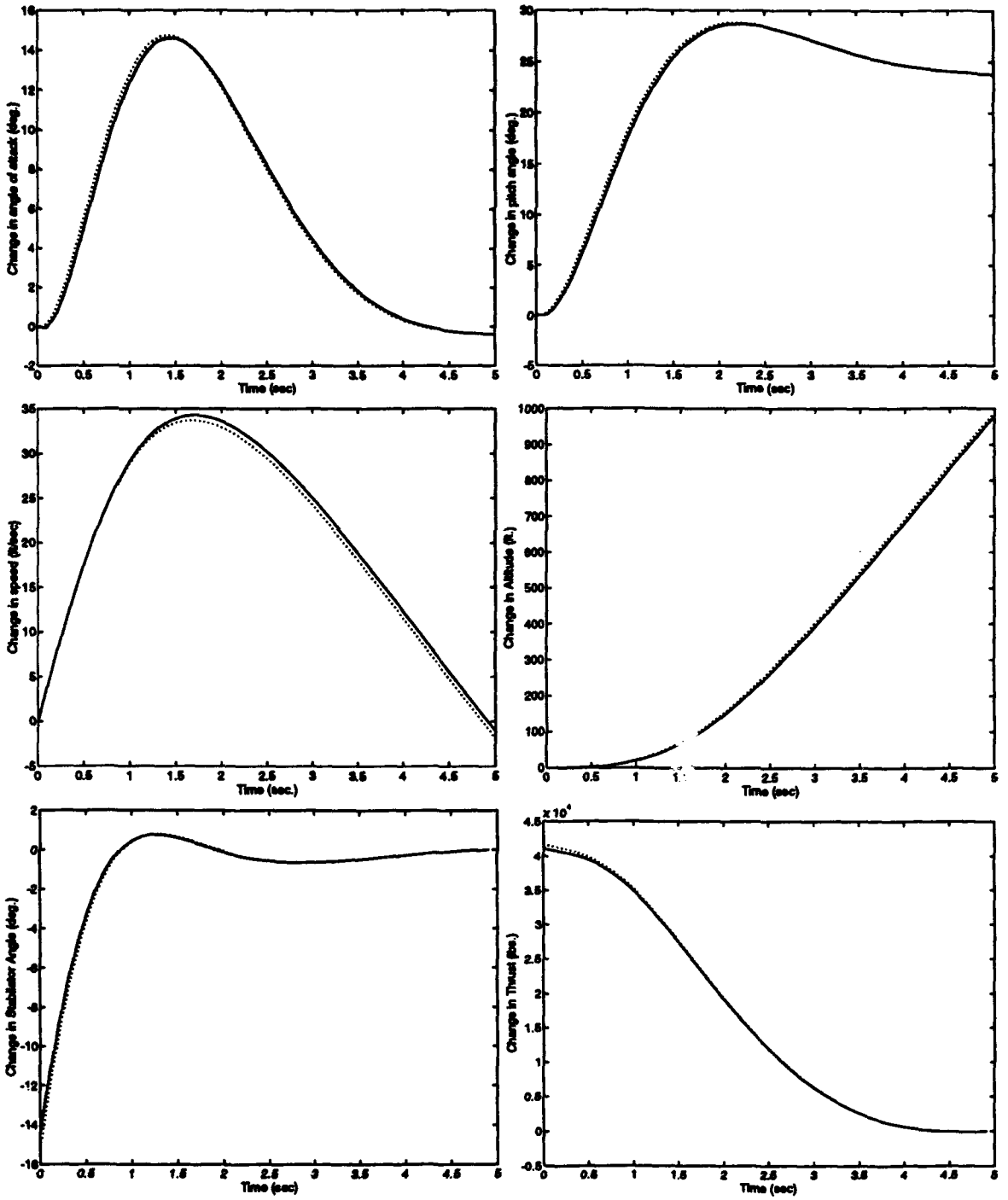


Figure 2.4.2.11: Optimal Trajectories using a Lagrange Multipliers Recurrent Network

2.4.3 Implementation of Control and State Limits

In many control applications, there may be physical limits on the values of the control or state variables. Simple inequality constraints can represent these, and can be implemented easily by constantly monitoring the state and control variables. If these variables are at the specified limits, and the rate of change of these variables is in the direction of exceeding these limits, then their rate of change is set to zero, and they are held at the specified limits. This is easy to do in hardware, by passing the controls or states through squashing functions, such as sigmoids, which limit the outputs of these controls and states to the desired values. The addition of the squashing functions will not alter the stability of the network, but it might result in a change of the rate of convergence.

If the linear inequality constraints are each a function of more than one control and/or state variables, it might be possible to implement these constraints by transforming them into equality constraints using slack variables.

2.4.4 Conclusion

In this chapter, we analyzed three different methods for finding the optimal control trajectories for linear systems, given an objective function. We tested the different methods on a linearized aircraft-model of longitudinal dynamics. A summary of the properties of the different techniques is shown in table 2.4.4-1 . From this table, Lagrange multiplier method satifies most of the desired objective criteria.

Table 2.4.4-1: Comparison Between Three Different Recurrent Neural Network Implementations Of Optimal Control

	Penalty Method	Gradient Projection	Lagrange Multipliers
Accuracy	Not Exact	Exact	Exact
Robustness	Robust	Not Robust	Robust
Relative Network Size	Small	Small	Large
Network Connectivity	Sparse	Dense	Sparse
Convergence Properties	Slow	Fast	Fast
W Matrix Computation	Simple mapping of A, B, Q, R, K	Matrix Inversion	Simple mapping of A, B, Q, R

3. PARAMETER IDENTIFICATION

The real-time optimal controllers, developed in the previous chapter, require an adaptive real-time parameter estimation module. The role of the parameter estimation module is to provide the controllers with fast and accurate estimates of the linearized system matrices, by measuring the input/output history of the different state and control variables of the aircraft. We can look at the parameter ID problem as a mapping from input-output data histories to unknown parameters, defined with respect to a particular class of models, for instance linear state space systems. Techniques for parameter estimation typically involve finding the parameters that optimize some objective criteria: such as minimizing the error between the measured and predicted outputs, or maximizing the likelihood that the input-output data result from the proposed model. More recently, there has been an increasing interest in state space parameter estimation using subspace model identification techniques. State space subspace system identification (S4ID) methods use linear algebra tools, such as QR and Singular Value Decomposition, to find the subspace that best fits the input-output data. If we have only input-output data measurements, the definition of states is not unique.

In this chapter, we will explore the use of different neural network architectures for the implementation of real-time parameter identification techniques. Neural networks offer many desirable features that might be useful for parameter identification. These features include :

- Fast optimization of linear and nonlinear objective functions: Hopfield-type neural networks, discussed in the previous chapter, possess such a capability. This feature can be used to implement optimization-based system identification algorithms in real-time. In a more indirect way, these networks are also capable of implementing S4ID estimation algorithms, by performing many of the matrix computations involved in implementing S4ID algorithms such as computing QR and Singular Value Decomposition.

- Ability to store knowledge and previous experience about the dynamic system. This feature may turn out to be particularly important in data-poor environments, where the states are either unavailable for measurements or highly corrupted with noise. The availability of previous knowledge may reduce the amount of data required to accurately estimate the parameters. For example, in normal aircraft operation, a content addressable memory can be trained to generate the jacobian matrices of the dynamic system, given only current measurements of the state and control vectors. This is not possible with conventional parameter estimation techniques, where usually the history of the state and control variables are needed for a robust estimation. In addition the control signal should be persistently exciting. Of course, since the memory-based network bases its decision only on the current

state and control variable and its previous experience about the behavior of the dynamic system in similar conditions, it may not detect a fast change in the dynamics of the system. Therefore, the memory-based network can augment a conventional parameter estimation technique which observes both state and control histories but cannot substitute it completely. Both modules can work in parallel, with the conventional parameter estimation technique providing continuous training for the memory-based network.

3.1 Previous Neural Network Approaches to Parameter Estimation

There has been relatively little published research on using neural networks for the estimation of parameters of dynamic systems. This may be due to the success of conventional parameter estimation techniques and the relatively small added value that neural networks may offer. Wang and Mendel (1991) trained a structured network to estimate the parameters of a moving average (MA) process using higher order statistics of the outputs. They also proposed techniques for extending their technique to autoregressive moving average (ARMA) processes. Other researchers have used limited histories of input/output data to train a feedforward network to estimate the parameters of a dynamic process (Samad and Mathur 1991; Foslien, Konar and Samad, 1992). One obvious problem with this approach is that the dimension of the feedforward neural network is very large for robust identification. Chu, Shoureshi and colleagues proposed using Hopfield networks for the identification of the parameters of linear systems in state-space form (Chu, Shoureshi and Healey, 1992); Chu and Shoureshi, 1992; Chu, Shoureshi and Tenorio, 1990).

3.2 Implementation of Optimization-Based Parameter ID Using Hopfield Networks

In this section, we describe optimization-based methods for the estimation of parameters of dynamic systems represented in state-space form. We then discuss how these methods can be implemented using Hopfield neural networks. We extend the work of Chu and Shoureshi to include the estimation of parameters in the case of known relationships between parameters. We will also discuss how to implement a forgetting factor for fast adaptation.

A Hopfield-based parameter identifier is illustrated in figure 3.2-1. The connectivity matrix encodes the input/output response history of the dynamic system, measurement error covariances and known parameter relationships. Initial parameter estimates, if available, help initialize the state of the network. We will describe now in detail how to synthesize the parameter ID Hopfield network to satisfy the above requirements.

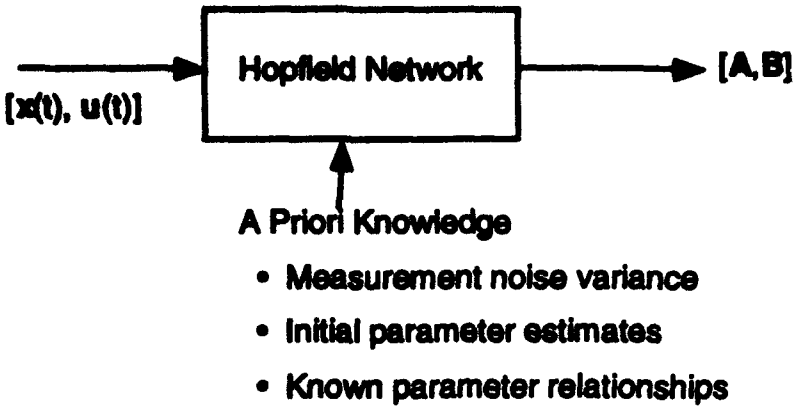


Figure 3.2-1: A Hopfield Network Parameter Identification Module for Linear Systems

3.2.1 Hopfield Parameter ID network

Given a particular model structure with unknown parameters, a parameter ID problem can be defined as finding the parameters which optimize a certain objective function such as the minimization of the error between model predictions and measured outputs, or the maximization of the likelihood that the unknown parameters produced the measured data. As we demonstrated in the previous chapters, recurrent networks may solve such optimization problems. We present here the derivation of a Hopfield parameter ID recurrent network for linear systems and discuss its possible advantages and disadvantages.

The system state equations for a discretized linear system can be defined as:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{3.2.1-1}$$

Our objective is to find the matrices **A** and **B** that, given $\mathbf{x}_k$ and $\mathbf{u}_k$, can predict, in a least square sense, the output $\mathbf{x}_{k+1}$. If we define the prediction error

$$\mathbf{e}_k = \mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k - \mathbf{B}\mathbf{u}_k \tag{3.2.1-2}$$

where **e** is an N-dimensional vector, and define an energy function

$$E = \frac{1}{2} \mathrm{Tr} \sum_{k=0}^{N-1} \mathbf{e}_k \mathbf{e}_k^T \tag{3.2.1-3}$$

It is easy to prove that the matrix differential equations :

$$\frac{dA}{dt} = -\epsilon \frac{\partial E}{\partial A} \tag{3.2.1-4}$$

$$\frac{dB}{dt} = -\epsilon \frac{\partial E}{\partial B} \qquad (3.2.1\text{-}5)$$

have an equilibrium point at the minimum of the energy function, therefore the matrices A and B converge to the optimal least squared error solution.

Substituting for the energy function from equation (3.2.1-3) above, the differential equations (3.2.1-4), (3.2.1-5) can be written as :

$$\begin{bmatrix} \frac{dA^T}{dt} \\ \frac{dB^T}{dt} \end{bmatrix} = -\epsilon \underbrace{\begin{bmatrix} \sum_{k=0}^{N-1} x_k x_k^T & \sum_{k=0}^{N-1} x_k u_k^T \\ \sum_{k=0}^{N-1} u_k x_k^T & \sum_{k=0}^{N-1} u_k u_k^T \end{bmatrix}}_{\text{Connectivity matrix W}} \begin{bmatrix} A^T \\ B^T \end{bmatrix} - \underbrace{\begin{bmatrix} \sum_{k=0}^{N-1} x_k x_{k+1}^T \\ \sum_{k=0}^{N-1} u_k x_{k+1}^T \end{bmatrix}}_{\text{Bias Matrix S}} \qquad (3.2.1\text{-}6)$$

where N is the number of data points in the measurement window. The dimension of the connectivity matrix W is (n+m) x (n+m), where n is the state dimension and m is the control dimension. The bias matrix S is of dimension (n+m) x n. The number of units in the Hopfield network representing the above system of equations is the same as the dimension of S. Monotonically increasing functions may be added to the above system of equations, but they are not necessary. It is important to realize that the connectivity matrix W is not constant and is a function of the correlations between input and output data. This makes it more difficult to implement in hardware.

The above parameter ID Hopfield network may be extended in different ways to obtain better estimates of the parameters as described below:

A Weighted Least Square Estimation

For nonlinear and time varying systems, the system parameters change with time. To track these time-varying parameters, more recent data are considered to be more relevant and given more weight. The weighting parameter, also called forgetting factor, can be easily incorporated into the connectivity matrices and bias terms of the Hopfield network, as shown by equation (3.2.1-7).

$$W = \begin{bmatrix} \sum_{k=0}^{N-1} \lambda_k x_k x_k^T & \sum_{k=0}^{N-1} \lambda_k x_k u_k^T \\ \sum_{k=0}^{N-1} \lambda_k u_k x_k^T & \sum_{k=0}^{N-1} \lambda_k u_k u_k^T \end{bmatrix} \qquad S = \begin{bmatrix} \sum_{k=0}^{N-1} \lambda_k x_k x_{k+1}^T \\ \sum_{k=0}^{N-1} \lambda_k u_k x_{k+1}^T \end{bmatrix} \qquad (3.2.1\text{-}7)$$

where λ_k is the forgetting factor, usually chosen to be $\lambda_k = \alpha^{N-k}$, with $0 < \alpha < 1$

Identification Of Parameters With Equality Constraints

As pointed out by Chandler, Pachter and Mears (1993) the exploitation of a priori information about relationships between the different parameters to be identified reduces the uncertainty in the estimation of these parameters. As mentioned in the previous chapter, it is possible to implement linear constraints in a Hopfield network using Lagrange multipliers.

3.2.2 Practical Issues In The Implementation Of State-Space Parameter ID Using Hopfield Networks

Since the Hopfield network is an implementation of the classical optimization based parameter ID methods, it shares many of the practical issues associated with these techniques, such as the requirement of persistently exciting inputs and the ability to stably track the parameters of a nonlinear or time-varying system. In addition, there are other problems that are due to the Hopfield implementation. One major problem is the possibility of ill-conditioning of the matrix W. This is can be remedied by scaling the different input and state variables, or equivalently, by using different time constants for the different parameters.

3.2.3 Flight Control Example

In this example, we used a Hopfield network to identify the parameters of the linearized longitudinal dynamics model of the F/A-18 aircraft at an altitude of 30,000 ft and a speed of 700 ft/sec. We used the optimal control inputs that were generated using the Lagrange multipliers Hopfield network optimal controller. A window of 100 data points at a sampling rate of 20Hz was used in the identification. We added white noise to the state variables with a magnitude equal to 10% of their variance. We used the parameters estimated using the Hopfield network and the control inputs to reconstruct the states. Figure 3.2.3-1 shows a comparison between the noisy state measurements and the state trajectories generated using the estimated parameters. As shown in the figure the Hopfield network was able to reproduce the states with a *high accuracy*.

In the above example, in order to make the Hopfield network work, we had to scale the altitude variable. Without scaling, the connectivity matrix becomes extremely ill-conditioned and requires a very small integration step.

Although Hopfield type neural networks are capable of performing state space parameter identification, as shown in the above example, they may not have any advantage over current techniques for small systems.

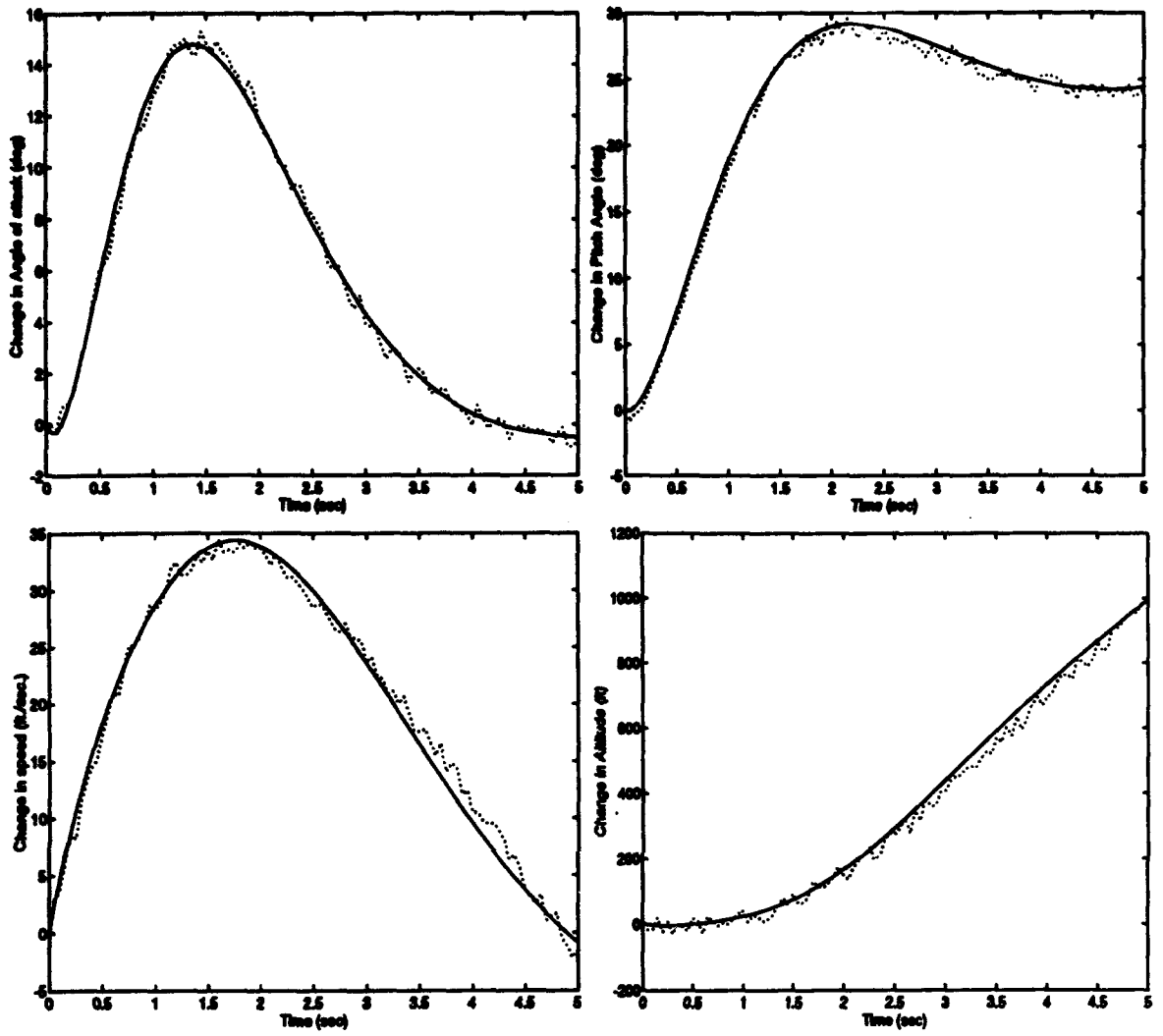


Figure 3.2.3-1: Noisy State Measurements (Dotted Lines) And Estimated States Solid Using Hopfield Parameter ID

4. LEARNING THE INVERSE DYNAMICS

We investigate in this chapter learning the inverse dynamics for use in computing the dynamic trim for aircraft FCS applications.

Problem Formulation

The inverse dynamics of a plant defined by the state equations (4-1) :

$$\frac{dx}{dt} = f(x,u) \tag{4-1}$$

where $x \in \Re^n$ and $u \in \Re^m$ can be defined as:

$$u_i = f_i^{-1}(x, \frac{dx}{dt}, u_{j \neq i}) \qquad i = 1, ..., m \tag{4-2}$$

where $u_{j \neq i}$ is an $\Re^{m-1}$ vector formed by excluding the control variable u_i from the control vector u. In general, the inverse $f_i^{-1}(\cdot)$ may not exist. If the control variables represent a nonredundant system, that is there is a unique control vector u which produces a given desired rate of change of the states $\dot{x}_d$, then the dependence of each control variables on the other control variables may be dropped and the inverse equations may be written as shown in equation (4-3):

$$u = g(x, \frac{dx}{dt}) \tag{4-3}$$

However, it must be noted here that the inverse function $g(\cdot)$ is not defined everywhere. It is only defined in a small submanifold $\Gamma \subset \Re^{2n}$. The submanifold Γ defines the set of reachable states and achievable rates of change of the states. For example, for a linear system, the achievable rates of change of the states from a particular state are defined by the hyperplane:

$$\dot{x} = Ax + Bu \tag{4-4}$$

For example if the system is a second order and the B matrix = $[0\ 1]^T$ and $x = [0\ 0]^T$ then the vector $\frac{dx}{dt}$ can only be in the direction of $B = [0\ 1]^T$. In such a case the inverse function $u = g(x, \frac{dx}{dt})$ at $x = [0\ 0]^T$ and $\dot{x} = [0\ \alpha]^T$ is not defined. If the number of control variables m is equal to the order of the system n, the columns of the matrix B are all independent and there are no limits on the control variables then the submanifold $\Gamma = \Re^{2n}$. Obviously these conditions are too restrictive and are seldom met in practice. Therefore, it is important to define the submanifold where the inverse is valid. Since the desired rates of change of the states do not, in general, coincide with the submanifold where the inverse is defined, it is important to find a suitable approximation to the inverse function $g(\cdot)$ in the region where such an inverse does not exist. This can be done either during the training of a neural network to perform the inverse or even during run

time. For example, at run time, one possible method to approximate the inverse outside the manifold where it is defined is to project the desired rates of change of the states into the manifold Γ and then use the inverse model. There is another approach that can be used to overcome the problem that the dimensionality of the rates of change of the states is usually higher than the dimensionality of the available controls: to define the inverse function $\mathbf{u} \equiv \mathbf{g}(\mathbf{x},\mathbf{v})$, where the vector $\mathbf{v}$ is of the same dimension as the control vector $\mathbf{u}$. We can then project the desired rates of change of the states to the variable $\mathbf{v}$, using for example a gain matrix $\mathbf{K}$ (e.g. $\mathbf{v} = \mathbf{K}\,\dot{\mathbf{x}}$).

Methods for training neural networks to perform the inverse function will be discussed in detail in the next section.

4.1 The Inverse Model as a Controller

In general, control system design can be viewed as an attempt to produce a well behaved inverse of the plant dynamics that has desirable robustness, stability and dynamic response characteristics. An exact inverse model of the dynamics is not always desirable if it does not possess good dynamic characteristics. In this chapter we discuss different techniques for building an adaptive neural network model of the inverse dynamics. We develop one such network for modeling the inverse longitudinal dynamics of a nonlinear F/A-18 simulator. We discuss methods for overcoming many of the shortcomings of inverse dynamics control.

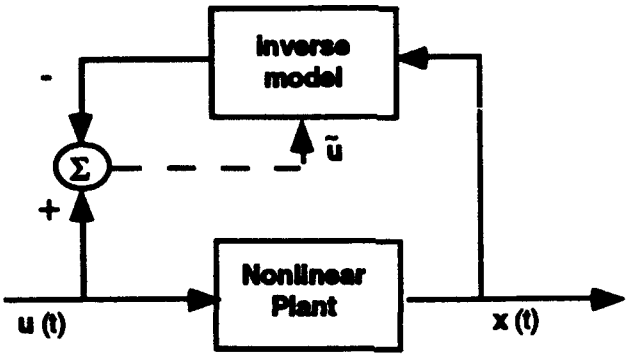
There are many different ways to train a neural network to produce the inverse of a dynamic system. Three of these ways are shown schematically in figure 4.1-1. Kawato and Gomi (1992) summarize the advantages and disadvantages of each of these three techniques. The simplest possible scheme for acquiring an inverse is what is called the direct inverse modeling (figure 4.1-1.a). In this scheme, the inverse model observes the realized trajectory of the plant and attempts to estimate the control command that generated that trajectory. The error between the inverse model estimate and the actual control command is used to train the inverse model. This scheme has been successfully applied by Atkeson (1989) to train a neural network to estimate the torques necessary to drive a robot arm along a desired trajectory. However, this technique by itself does not solve the problems associated with inverse modeling discussed in the previous section, namely that it will fail when the inverse is not unique. This problem can be solved by adding extra constraints which make the network choose only a unique inverse. Also, this technique will fail when the desired state change cannot be achieved, since the neural network will not have any training examples in that region. One solution in such cases might be to use the control values of the nearest experiences.

The second approach to learning the inverse dynamics, developed and used by Jordan (Jordan, 1989); (Jordan and Rumelhart, 1992), is to first transform errors in the states $\tilde{\mathbf{x}}$ to errors

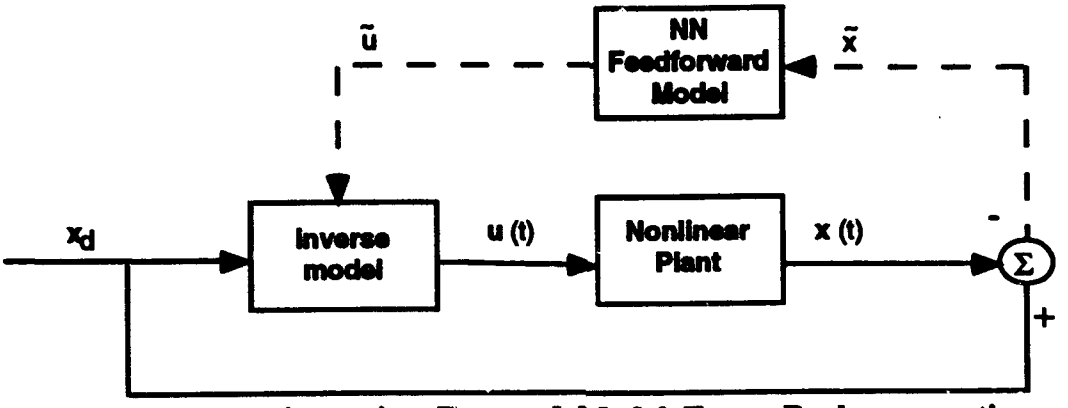
in control $\mathbf{u}$ by propagating $\mathbf{x}$ backward through a forward model of the plant dynamics, as shown in figure 4.1-1.b. This approach solves the problem of finding the control values when the inverse is not unique, since the objective function to be minimized in this case is the output error. This is unlike the direct inverse, where the learning objective is to minimize the error in the estimated control. Moreover, it is possible to add a regularization term to the backpropagation algorithm to achieve better control qualities. For example the objective function in training the inverse model for a discrete dynamic system can be expressed as shown in equation (4.1-1) below:

$$\min_{\mathbf{u}} \left(FN(\mathbf{x}_k,\mathbf{u}_k,W) - \mathbf{y}_k\right)^T Q \left(FN(\mathbf{x}_k,\mathbf{u}_k,W) - \mathbf{y}_k\right) + \mathbf{u}_k^T R \mathbf{u}_k \qquad (4.1\text{-}1)$$

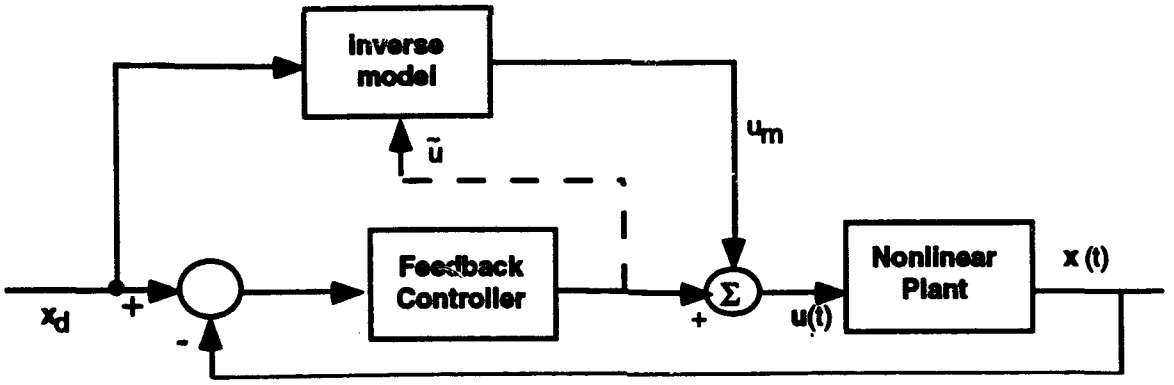
where $FN(\mathbf{x}_k,\mathbf{u}_k,W)$ represents the neural network forward model, $\mathbf{y}_k$ is the desired output vector for training point k, and $\mathbf{x}_k$ is the output of the plant resulting from applying control $\mathbf{u}_k$. The matrices Q and R are weighting matrices.



a. Direct Inverse Learning



b. Inverse Learning using Forward Model Error Backpropagation



c. Inverse Learning Using Feedback Error

Figure 4.1-1. Different Techniques for Learning the Inverse Model

W represents the parameters of the feedforward neural network. The change in the control $\tilde{u}_k$ for updating the value of u_k can be found using any of the nonlinear optimization techniques. For example using gradient descent this update rule can be written as:

$$\tilde{u}_k = -\frac{\partial FN(x_k,u_k,W)}{\partial u_k} Q\,\tilde{x} + Ru \qquad (4.1\text{-}2)$$

The update rule for the inverse model neural network parameters can then be derived :

$$\Delta V = \left(\frac{IM(x_k, x_{k+1}, V)}{V}\right) \tilde{u}_k \qquad (4.1\text{-}3)$$

where $IM(x_k, x_{k+1}, V)$ represents the inverse model and V represents the inverse-model neural network parameters. One of the advantages of the backpropagation through the forward model technique is that it is possible to make the inverse more sensitive to a particular state through the choice of the Q matrix. For example if the measurements of a particular state are very noisy, we can reduce the sensitivity of the inverse with respect to that particular state by reducing the corresponding values of the Q matrix. It is also possible to regulate the degree of utilization of the different controls through a proper choice of the matrix R. Of course, the major disadvantage for the backpropagation through the forward model technique is that it requires building a forward model of the dynamic system in addition to the inverse model. An approach similar to the feedforward backpropagation has been suggested by few researchers. In this approach, only the forward model is learned and adapted to changes in plant dynamics. Then using nonlinear root-finding techniques, similar to those used in aircraft trimming, it may be possible to find the controls that produce the desired response. The obvious disadvantage with this approach is that it requires real-time solution of coupled nonlinear equations, which may be computationally expensive.

The third approach to learning an inverse model, shown in figure 4.1-1.c and called feedback error learning, is described by Kawato and Gomi (1992). In this technique, the output of an error feedback controller is used as an error signal to train the inverse model. It is important to note that even though the feedback controller may be linear, the neural network inverse model is, in general, non linear after training. However, the learned inverse model still depends on the quality of the feedback controller and its preferences. For instance if the feedback controller consists of high feedback gains corresponding to unstable states and lower gains for originally stable states, the resultant inverse model will also reflect the same priorities. The same can also be said for the relative utilization of the different control variables.

4.2 Learning the Inverse Longitudinal Trim Function of the F/A-18

A neural network can be trained to produce the control surface commands to trim the aircraft at given flight conditions and aircraft trim states. The inverse trim network is a specialized model of the full inverse dynamics, where the desired rates of change of the states are all set to zero. This reduces the input space of the inverse trim network considerably and makes it much easier to train with fewer training examples. Although trimming the aircraft can be done using

conventional numerical-trim-finding algorithms, the advantage of using a neural network approach is the possible adaptation to aircraft dynamics.

We implemented an inverse trim neural network of the longitudinal dynamics of the F/A-18 aircraft. The inverse trim neural network consisted of two inputs and three outputs. It took as input the desired altitude and speed of the aircraft and produced as output the elevator angle (δ_e), the total thrust and the angle of attack (α) required to trim the aircraft.

The first step to train the inverse trim neural network was to generate a trim database. We used a numerical-trim-finding algorithm to generate the database. The training set contained 200 data points. Each data point consisted of the trim quintuple (speed V_T, altitude h_T, angle of attack α_T, elevator angle δe_T and the thrust thr_T). The values of V_T and h_T were randomly and uniformly chosen to be in the range of [300-900 ft./sec] and [0 - 60,000 ft.] respectively. The trim condition was for a level flight, therefore the pitch angle θ was set equal to the angle of attack α and the pitch rate q is equal to zero. Therefore we did not need to include the value of θ and q in the computation of the inverse trim model.

The neural network model for the inverse trim consisted of three HyperBF networks with Gaussian basis functions of variable widths. Each HyperBF network computed one of the trim variables α_T, δ_{eT} and δ_{thrT}. The HyperBF network structure and the training algorithms used to compute the network parameters are presented in detail in Appendix A. Each network contained 80 basis functions, with centers randomly and uniformly distributed in the range of the training set. For each network, the parameters to be identified consisted of the 80 linear coefficients C_i and 2 parameters for a diagonal weight matrix W (see Appendix A for an explanation of the meaning of the parameters). We used a least square error minimization between the estimated and exact output values as the criteria for choosing the network parameters. The relative rms error defined as:

$$\text{Erms} = \frac{\sqrt{\sum_{i=1}^{200} \left(x_{Ti} - x_{ei}\right)^2}}{\sqrt{\sum_{i=1}^{200} x_{Ti}^{\,2}}} \qquad (4.2\text{-}1)$$

was used to evaluate the performance on the training set, where x_{Ti} represents the exact trim value and x_{ei} is the estimated value and 200 is the number of data points in the training set. This error was found to be less than 0.01 for all the three variables α_T, δ_{eT} and δ_{thrT}.

To test the performance of the inverse trim network, we compared its output with the exact output generated using a numerical trim solver. The data used in the test set were different from those used in training the inverse trim networks. Figure 4.2-1 compares the exact outputs with the outputs generated using the inverse trim networks at different trim states. The results shown are for

trimming the aircraft at different altitudes and at a constant trim speed v_T of 700 ft/sec. As shown in the figures, the estimated trim values match the numerically computed ones. The error is higher near the boundaries, where the amount of training data are not sufficient for exact generalization.

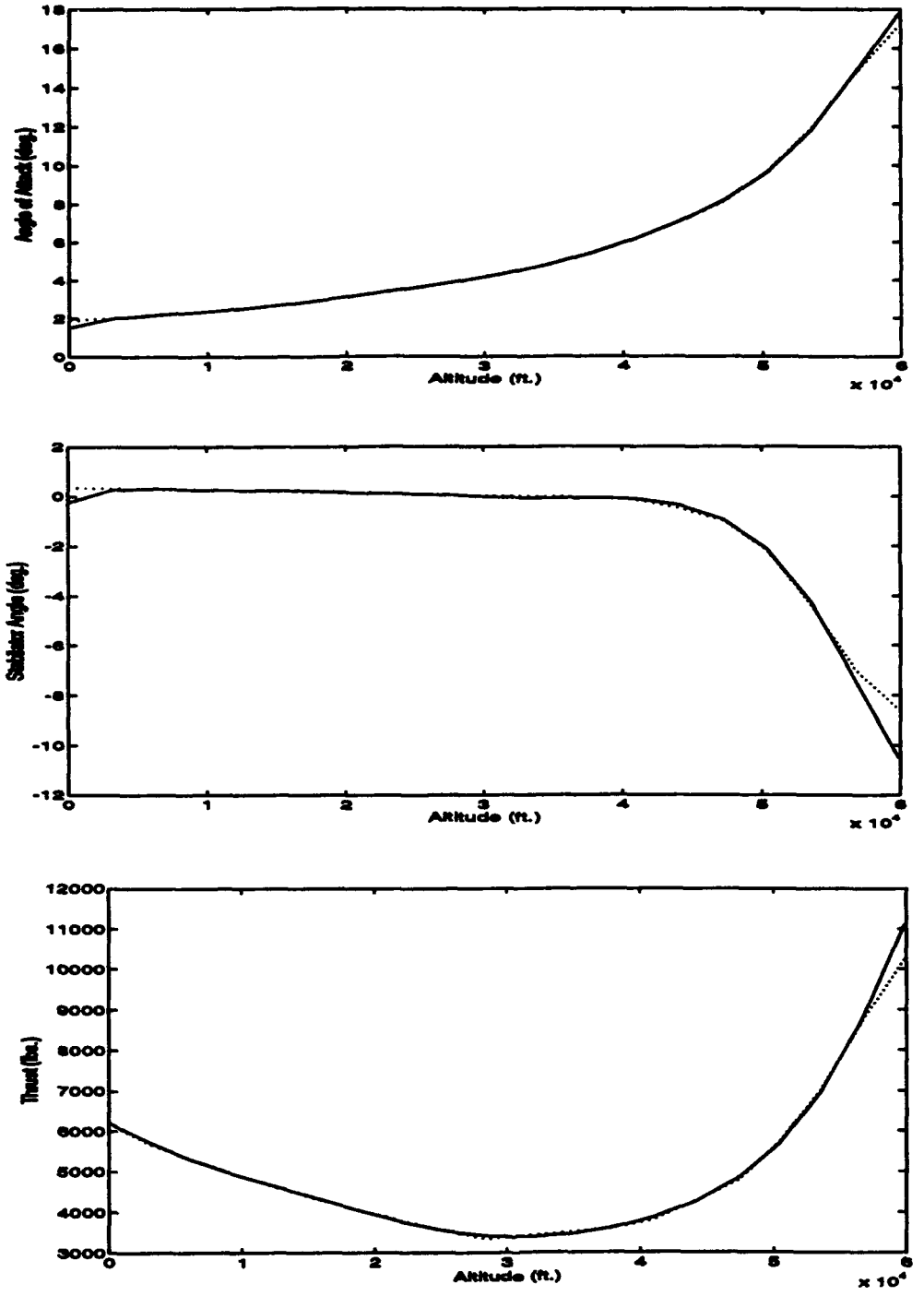


Figure 4.2-1. Comparison Between Neural Network Inverse Model For Trim (Solid Line) And Numerically Computed Trim Values (Dotted Line) At A Constant V_t = 700 Ft/Sec.

4.3 Learning the Inverse Longitudinal Dynamics of an F/A-18 Aircraft

Modeling the inverse trim function presented in the previous section is a specialized version of a full inverse model. Equation (4.3-1) defines a general inverse model for the longitudinal dynamics of an aircraft.

$$\delta_e = g_1(\alpha,\theta,h,v,q,\dot{\alpha},\dot{h},\dot{u},\dot{w},\dot{q})$$

$$\delta_{thr} = g_2(\alpha,\theta,h,v,q,\dot{\alpha},\dot{h},\dot{u},\dot{w},\dot{q}) \qquad (4.3\text{-}1)$$

The fact that we can choose all the states and their rates of change independently makes the input space much larger than in the case of the inverse trim model. For the inverse longitudinal dynamics model of the F/A-18 that we used, the input space was formed of five states and their rates of change, for a total of 10 dimensions, as shown in equation (4.3-1). There were two outputs, the stabilator angle and the thrust. We used two HyperBF networks with Gaussian units, one for each output. The Gaussian RBF units had variable widths along the different input dimensions. The HyperBF parameters were estimated using the heuristic method described in Appendix A, in which the widths of the Gaussians are function of the average sensitivity of the inverse model in the different directions, in addition to the range of the different variables.

4.3.1 Training the Gaussian HyperBF Inverse Model

We trained the two HyperBF networks using 2000 training points randomly and uniformly distributed in the input space. Each training point consisted of twelve dimensions, ten for the inputs and two for the output of the inverse model. The training points were generated using the direct inverse method described above. At each given random state, we applied a random input to the aircraft model and observed the resulting rate of change of the states. We then used the states and the rates of change of the states as inputs to the neural network inverse model and trained the network so that its output approximated the controls that produced the change in states. The network contained 250 Gaussian HyperBF functions, with their centers randomly and uniformly distributed in the same range as the input data We tested the resulting inverse model network on a different 2000 points. We used equation (4.2-1) to quantify the error in the estimation. The relative rms errors in estimating the stabilator angle and the thrust for both the training and test sets are shown in table 4.2-1 below:

Table 4.2-1. Relative Rms Error In The Estimation Of The Training And Test Sets Using A Neural Network Inverse Model.

	Stabilator Angle Error	Engine Thrust Error
Training Set	0.035	0.008
Test Set	0.061	0.0196

As shown in the table, although the error in estimating the test set is about twice as large the size of the error in estimating the training set, it is still small, given the very small number of training data relative to the input space. Figures 4.3-1 and 4.3-2 compare the correct outputs with the outputs computed using the inverse model network, for 50 points of the training and test sets respectively.

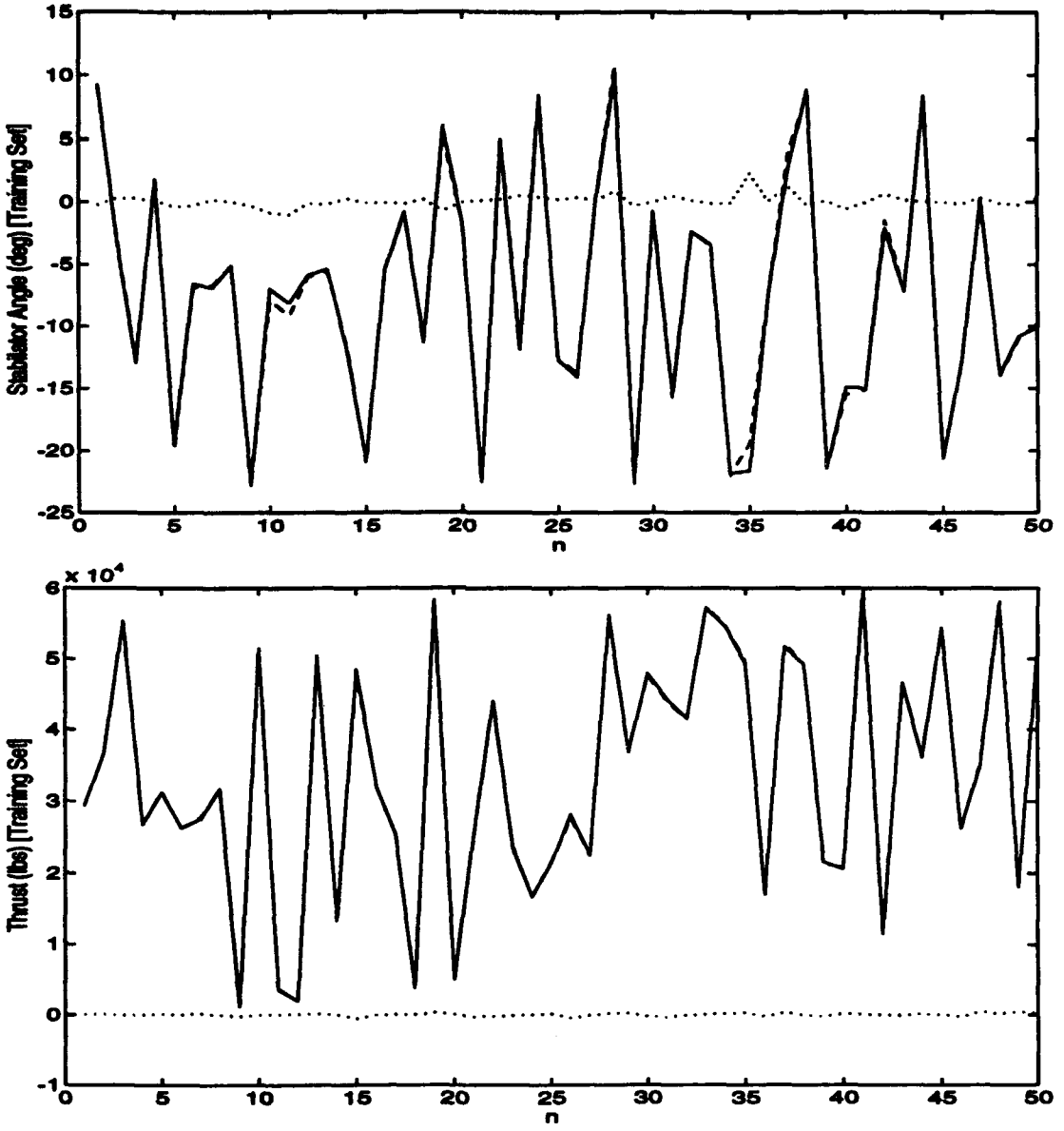


Figure 4.3-1: Performance Of The HyperBF Inverse Dynamics Model On 50 Points Of The Training Set. Dashed Lines Represent The Correct Values And Solid Lines Are The Estimated Values. Dotted Lines Represent The Errors.

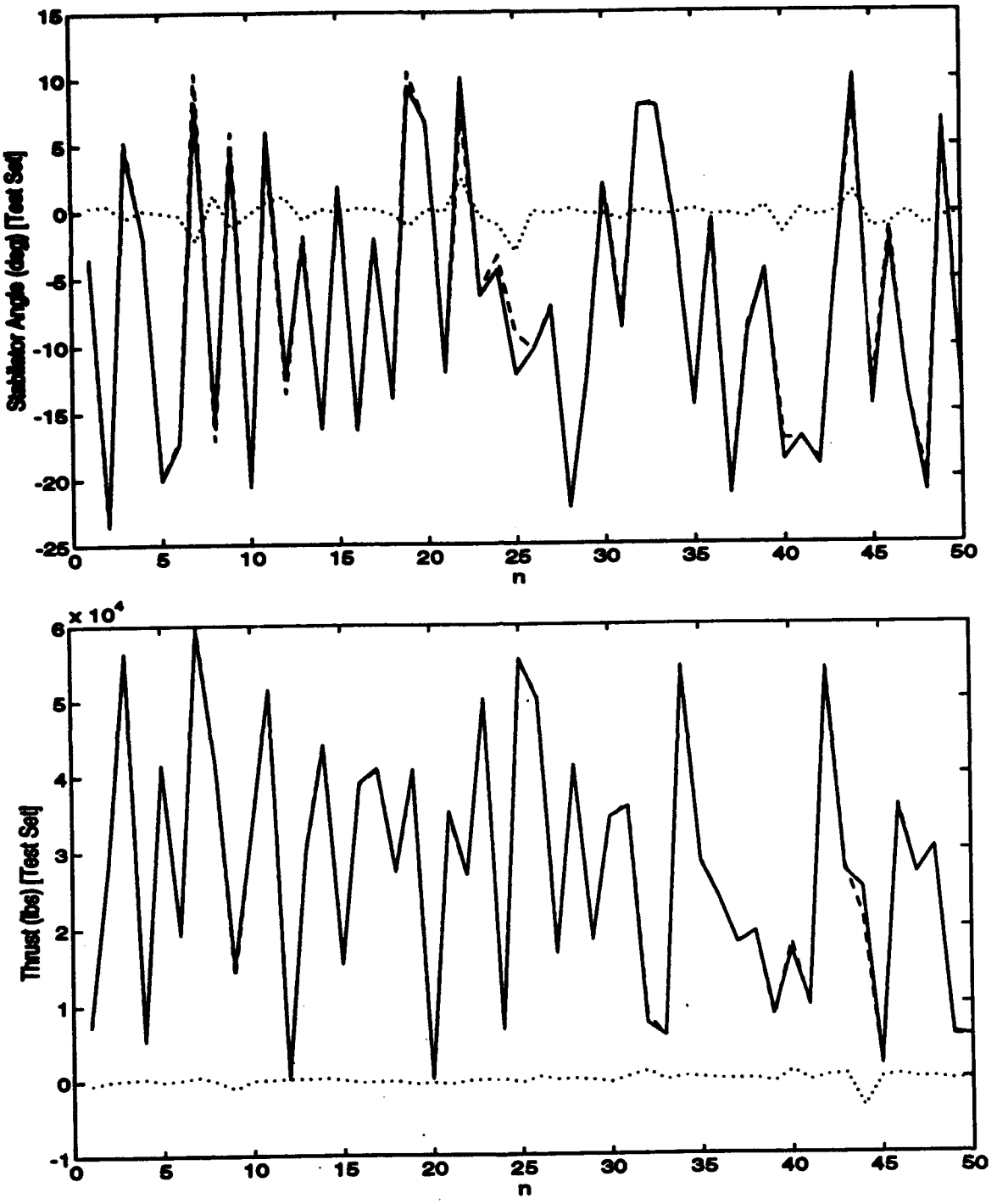


Figure 4.3-2: Performance Of The HyperBF Inverse Dynamics Model On 50 Points Of The Test Set. Dashed Lines Represent The Correct Values And Solid Lines Are The Estimated Values. Dotted Lines Represent The Errors.

4.4 Using the Direct Inverse Model Network to Control the F/A-18 Longitudinal Dynamics.

We tested the trained inverse model network for the control of the longitudinal dynamics of the F/A-18 aircraft. The control loop used is illustrated in figure 4.4-1.

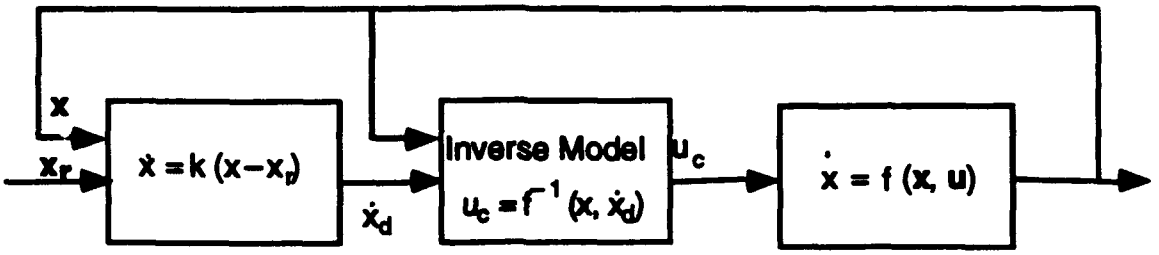


Figure 4.4-1. Control Loop To Test The Inverse Model Network

A desired state trajectory is generated off-line. The difference between the desired and actual trajectories are multiplied by a gain matrix (a diagonal matrix **K** in our case), to compute desired rates of change of the states. The actual states and the desired rates of change of the states form the inputs to the inverse model networks, which compute the controls to use to reach the desired trajectories.

When we used the direct inverse network generated using the above approach, the resulting control loop was unstable. This result was surprising given the very good approximation of the inverse model. However after a closer analysis, it was found that since the network was trained only to generate controls for feasible rates of change of the states, the inverse model network performance outside this region could not be predicted. This is the problem of the non-existence of the inverse discussed at the beginning of this chapter.

A Stable Inverse Model Network

To remedy the problem that the inverse may not exist, we have to provide the network with some training data in the region outside the feasible set of rates of change of the states. As discussed at the beginning of this chapter, we can use a criterion for selecting the control as a function of the current state. For example we can use feedback error learning Kawato and Gomi (1992) to train the network. We tried here an off-line approach to train the HyperBF networks. Part of the training examples were generated in the following way:

- Select random states and desired rates of change of the states within a specified range.
- Linearize the nonlinear aircraft model around the chosen state by computing the jacobian of the nonlinear dynamics at the current state using numerical methods. The linear aircraft model is described by equation (4.4-1)

$$\dot{x}_d = A\, x_r + B\, u \qquad (4.4\text{-}1)$$

where $\dot{x}_d$ represents the desired rate of change of the states and x_r is the chosen random state.

- Find the control vector u using the following equation :

$$u_r = g((B^T B)^{-1} B^T (\dot{x}_d - A\, x)) \qquad (4.4\text{-}2)$$

where the function g(.) limits the values of the controls to within the allowed region. For the simulations that follow we used a linear function g(.) with unity slope and with hard saturation at the limits of allowed controls.

- Use the set (x_r, $\dot{x}_d$, u_r) as a training example to train the inverse model neural network.

We used a training set of 2000 examples, with 400 examples generated using the above procedure. The remaining training examples were generated using the direct inverse approach, as before.

Results

We tested the inverse model HyperBF network generated using the above procedure for tracking three different trajectories. The control loop used is as illustrated in figure 4.4-2. Desired trajectories were generated as follows:

- Either a desired altitude or speed trajectory was first specified, the other variable was kept constant. We used a fifth order polynomial to generate a smooth desired trajectory.
- The angle of attacks required to trim the aircraft at the initial and final states were specified. The angle of attack trajectory was taken to be the linear interpolation between the two trim angle of attacks.
- The pitch angle and pitch rate were computed from the other three states to achieve the desired trajectory.

It is important to note that the desired trajectories may not have been achievable exactly for all the states, since we only used kinematic relations between the states to derive the trajectories. The desired rates of change of states were generated using a simple gain, multiplying the error between the current state and the desired current state as shown in equation (4.4-3).

$$\dot{x}_d(t) = K\,(x_d(t) - x(t)) \qquad (4.4\text{-}3)$$

K is assumed to be a diagonal matrix. At a sampling rate of 20Hz, the numerical values used for the gains are $k_h = 20$, $k_q = 5$, $k_{\dot{u}} = 10$, $k_{\dot{v}} = 10$; $k_{\dot{q}} = 10$, $k_{\dot{\alpha}} = 1$

We used these gains for the three examples that we describe below.

Example 1

The control objective in this example is to track a smooth increase in altitude from 20,000 ft to 40,000 ft in 250 sec. The speed is required to be constant at a value of 700 ft./sec. The angle of attack is a linear interpolation between the trim value at (20,000 ft / 700 ft/sec) and (30,000 ft./700 ft./sec.). The pitch angle is derived by first computing the flight path angle (γ) from the desired altitude and speed trajectories and then computing $\theta(t) = \alpha(t) + \gamma(t)$.

Using the gains mentioned above and the trained HyperBF inverse networks, the trajectories obtained for the different variables are plotted in figure 4.4-2. The dotted lines represent the desired ideal trajectories and the solid lines represent the simulated trajectories using the inverse dynamics HyperBF neural networks. As shown in the figures, the simulated trajectories are very close to the desired ones. The discontinuities shown in the figure are due to the nonlinear software simulation of the dynamics and not due to the controller.

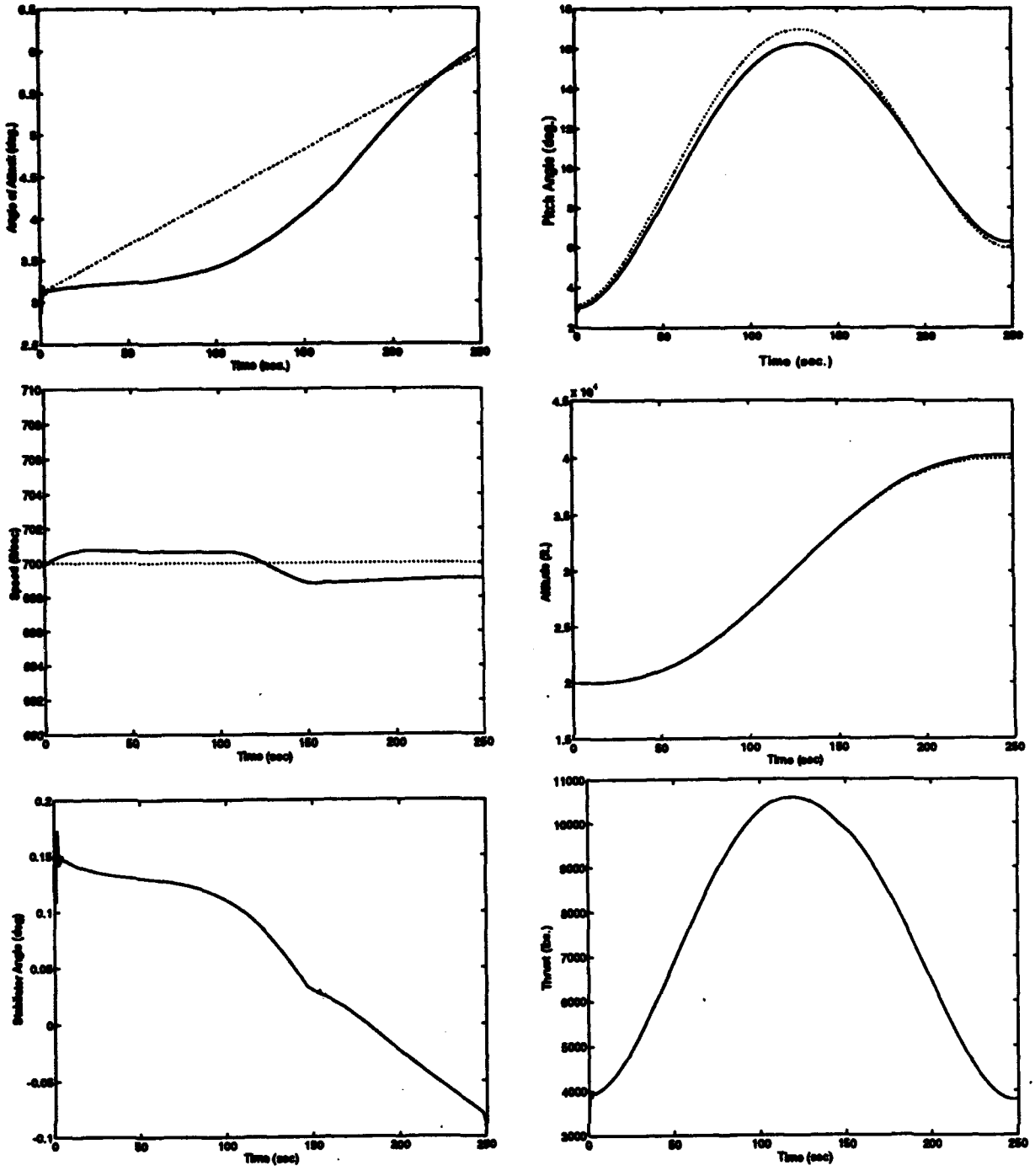


Figure 4.4-2: Comparison Between Desired (Dotted Line) And Simulated (Solid Line) Trajectories For Tracking An Altitude Command on Nonlinear Simulator

Example 2

In this example, we desire to track a speed change from 500 ft./sec. to 800 ft./sec. in 20 sec, while keeping the altitude constant at 30,000 ft. The desired and the simulated trajectories are

shown in figure 4.4-3. As shown in the figures, the simulated trajectories approach very well the desired one. Again the discontinuities in the different trajectories are due to discontinuities in the simulation itself and not the inverse model controller.

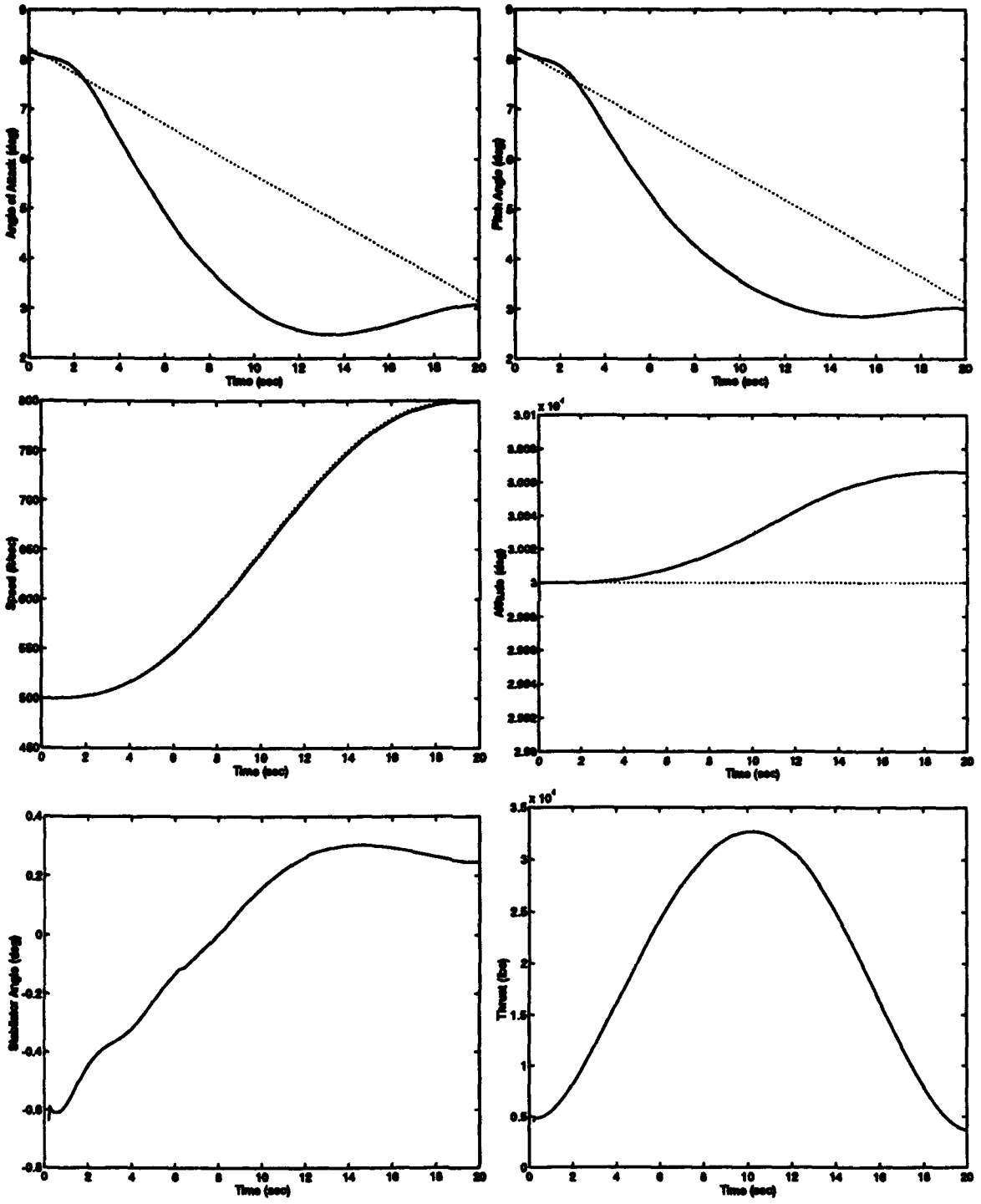


Figure 4.4-3: Comparison Between Desired (Dotted Line) And Simulated (Solid Line) Trajectories For Tracking A Slow Speed Command

Example 3

This example is similar to the previous one, except that in this simulation it is required to achieve the desired trajectory in 10 sec. only. This will test the inverse model controller near the limits of the controls. Again in this case, the performance was still satisfactory. The thrust computed using the inverse model HyperBF network exceeded the maximum values of 60,000 lbs. and had to be limited at this value for the simulation. This shows the ability of the trained HyperBF inverse dynamic model to extrapolate the controls necessary to achieve a high rate of change of the states (the network was only trained for controls up to 60,000 lbs.). The results of the simulation are shown in figure 4.3-4.

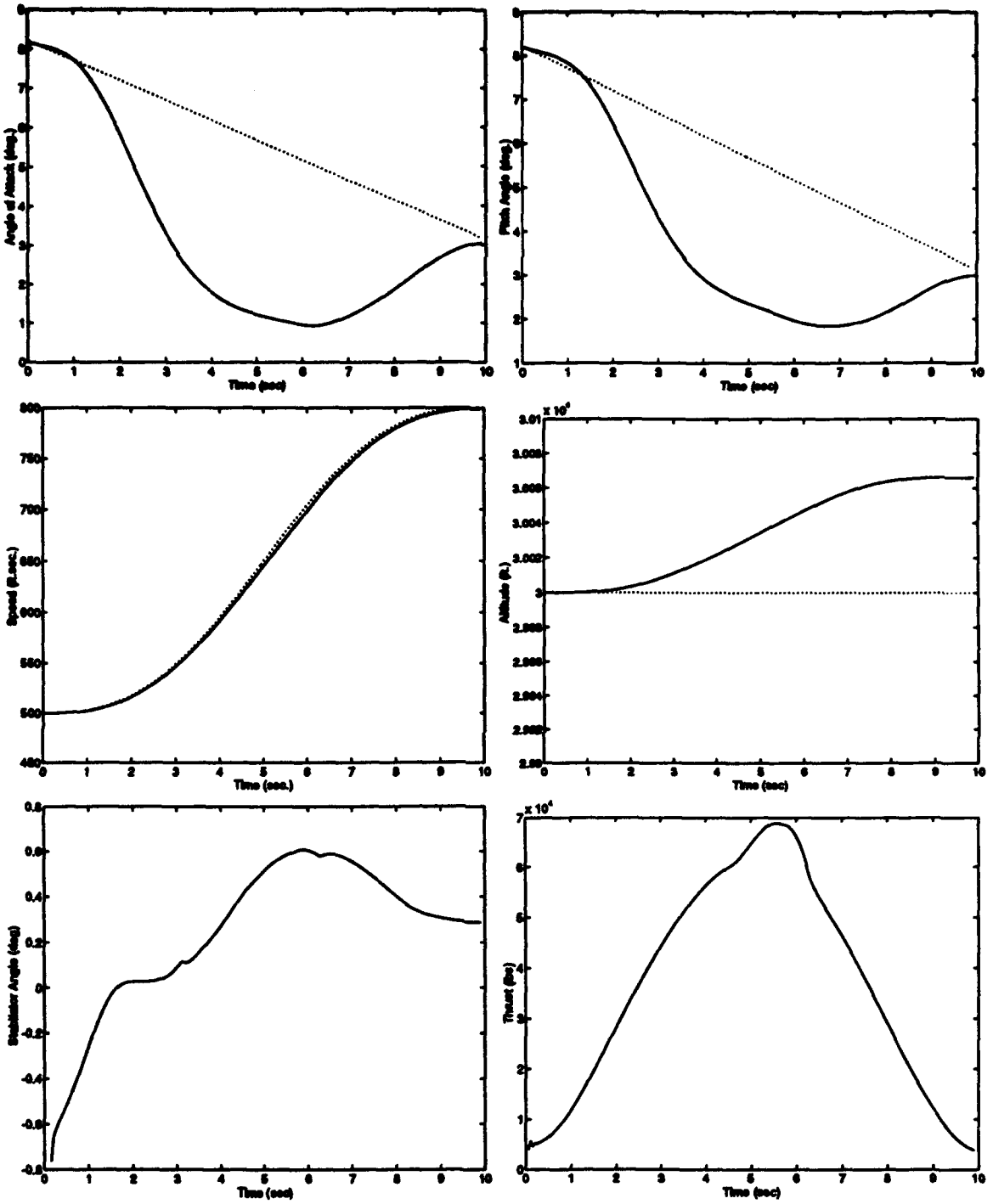


Figure 4.4-4: Comparison Between Desired And Simulated Trajectories For Tracking A Fast Speed Command

4.4 Summary and Conclusions

In this chapter we presented several procedures for generating an inverse dynamic model. We used HyperBF networks to model the longitudinal inverse dynamics of an F/A-18 simulator.

The HyperBF networks with Gaussian units were al le to generalize very well, given the extreme sparsity of the training examples. Theoretically, this is only possible if the inverse dynamics as a function of the states and the rates of change of the state is smooth. If the inverse dynamics are not smooth, it may be desirable to approximate it with a smoother function to achieve better and faster learning of the inverse function. Since the inverse dynamics do not exist everywhere, it is also necessary to provide the network with examples where the strict inverse does not exist. Therefore, the direct inverse model for training a neural network is not adequate and may result in unstable systems in most cases, if not augmented with an additional controller to generate controls when the desired trajectories are not achievable.

5. CONCLUSIONS AND PHASE II RECOMMENDATIONS

5.1 Phase I Contributions

In this phase I effort we investigated the feasibility and possible advantages of using different neural network architectures for performing different control functions. Our major contributions in this phase are the following:

- Studying the properties of the different techniques for implementing real-time optimal control using dynamic neural networks of the Hopfield-type.
- Developing and testing a *new* constraint satisfying dynamic neural network, based on Lagrange multipliers method, that is suitable for real-time adaptive optimal control.
- Proposing and testing different alternatives for implementing system parameter identification using different neural network architectures. We conclude that dynamic neural networks of the Hopfield type can perform parameter ID, but they *offer little advantage* over other real-time computational techniques for small size systems. We propose a parameter ID network based on associative memory, which is trained to estimate the network parameters based on the current state of the dynamical system. This network does not need persistence of excitation for adequate identification, but it is very *slow at* recognizing rapid changes in system dynamics. It can augment the performance of a conventional parameter ID system and help in failure detection.
- Studying different techniques for training and implementing an inverse dynamics neural network and investigating the advantages and disadvantages of these techniques.
- Implementing and testing a *novel* inverse trim based on HyperBF neural network.
- Successfully implementing an inverse dynamics model for the F/A-18 longitudinal dynamics using Gaussian HyperBF neural networks.
- Testing the HyperBF controller for the control of three different trajectories.

5.2 Conclusions

We believe that the biggest potential for neural networks in control is the exploitation of the ability to design adaptive nonlinear controllers. Based on the simulations performed in this phase I study, we show that Hopfield and RBF feedforward network architectures may have a great potential in the control of nonlinear systems. In particular, Hopfield implementation of Lagrange multiplier method is suitable for real-time adaptive optimal control. Similarly, RBF feedforward

neural network architectures are suitable for learning inverse dynamics and inverse trim in aircraft FCS applications. In addition, RBF feedforward are easier to train than backpropagation sigmoid networks since RBF formulation results in linear parameters.

The initial simulations we performed show very promising results as exemplified by the small control errors in closed-loop simulations using the nonlinear F/A-18 longtitudinal dynamics. Further studies are needed to test the applicability of the techniques to real world problems and to study the robustness, stability and general reliability of the proposed neural techniques. Neural networks by themselves cannot be the panacea to all the nonlinear control problems. An effort has to be made to incorporate all the available knowledge about the dynamic system to achieve good performance. In the next section we propose several extensions to the current effort to analyze and improve the performance of neural network controllers.

5.3 Phase II Recommendations :

We describe below several extensions to the research reported here which may help solve many of the problems associated with the current neural network approaches. Table 5.3-1 summarizes the recommendations we propose for a phase II study and relate these recommendations to what has been achieved in phase I. The proposed phase II recommendations are discussed in more detail in the next section.

Use of domain specific knowledge

In this phase I study, our main focus was the proof of the concept that neural networks can be used successfully in control. We mainly used brut force learning and in optimization techniques to prove our ideas. In phase II, we propose to use more specific knowledge about flight control in designing the architecture of the neural networks, in its training, and optimization. We believe that the inclusion of domain specific knowledge , such as relationships between variables or known effects of a particular input variable, can reduce the amount of training considerably and increase the accuracy of the generated neural networks.

Integration of the different modules

In the phase I effort, we tested the performance of each neural network module separately. However, we did not test the performance of the whole dynamic system. In Phase II, we propose to test the complete control system for performing large maneuvers. We will simulate situations such as failure to test the robustness and adaptation of the proposed techniques. We also propose to develop and incorporate a handling qualities model in the control loop.

Improving the robustness of the neural networks modules

Under the current study, we used ideal simulations with no noise to train and test the performance of the developed neural network modules. The performance of these systems may be different in real time applications where noise is present. In phase II, we propose to study the robustness of the neural network modules to noise and to explore different methods to improve their performance when noise is present. Methods that we plan to explore include: artificially inducing noise in training, using domain specific knowledge, exploring different architectures that are more immune to noise, and adding a noise robustness term in the objective function used to train the neural networks.

On-line design of optimal controllers

Many of the techniques used in designing control systems can be formulated as the optimization of an objective function. Techniques such as pole placement, LQG, loop transfer recovery, H_∞ design and μ–synthesis are examples of such techniques. In the current study, we explored different methods for optimization using dynamic neural networks. We propose to extend these methods to the on-line design of controllers that can be formulated as the optimization of an objective function.

Using bi-directional associative memories in control

Bi-directional associative memories (BAM) do not classify the different variables as inputs or outputs to the network. Any variable can be either an input or an output variable. The role of the BAM network is to predict the values of the missing variables. BAM networks may prove to be very useful in applications such as an inverse trim network, where the variables specified vary depending on the control objective. In phase II, we plan to explore methods for implementing BAM networks and explore their use in control systems.

Reducing the dimensionality of the inverse dynamics network:

In phase I, we developed full inverse dynamics models. This resulted in a high-dimensional neural network that requires a large number of data points for training. We also have shown that there is a smaller dimensional space where the inverse is properly defined. Moreover, it may be unrealistic to specify the desired rates of change for all states of the dynamic system. In phase II, we plan to explore methods for reducing the number of input dimensions in the inverse model.

On-line learning of the inverse function

In the current work, the training of the inverse dynamics was done off-line and then the trained inverse model was tested using the nonlinear longitudinal flight simulator. No learning occurred in real-time. In phase II, we propose to implement continuous training of the inverse model. Moreover, we propose to test the real-time adaptability of the inverse model network by varying the system dynamics. We plan to compare both Kawato's feedback error learning and Jordan's back-propagation through a forward model.

Optimization of optimal control and identification

Many control and identification problems are formulated as the optimization of an objective function. However, the choice of the objective function itself relies on heuristics and trial-and-error procedures. In phase I, we assumed these objective functions to be completely specified by the control system designer. In a phase II study, we plan to explore ways for automating the choice of the objective functions based on more abstract goals. We plan to explore fuzzy logic and neural network techniques which map the abstract goals into objective function parameters and weights. Moreover, we propose to explore neural network techniques for multi-objective optimization. Multi-objective optimization may be useful when there are conflicting goals to optimize, such as the case in the control-identification tradeoff discussed in the parameter ID chapter.

Different neural network implementations of system ID

In phase I, we only tested the implementation of state space models system ID using Hopfield optimization networks. In a phase II effort, we propose to implement different approaches to parameter ID. In particular, we plan to implement S4ID using Hopfield optimization networks and implement parameter ID based on associative memories.

Expanded aircraft simulations

In the current effort, we limited our simulations to the longitudinal dynamics of an F/A-18 fighter aircraft. Moreover, the maneuvers we simulated were limited to the available nonlinear flight simulator capabilities. Under a phase II study, we plan to acquire a more detailed flight simulator, expand our simulations to the aircraft full dynamics, and simulate a wider range of maneuvers. This will allow us to test our neuro-controllers under a wide variety of situations and will allow us to study the limitations of the techniques we developed.

Software implementation of neuro-control systems :

The design and implementation of neuro-control modules is a time consuming and expensive process. We propose to use the experience we gained in Phase I to develop a neuro-control toolbox with advanced user interface, based on our CASYS software design tool.

Table 5.3-1 Features Of Phase I And Phase II Efforts

Feature	Phase I	Phase II
Objective	Demonstrate applicability of neural network controllers	Produce a neuro-control toolbox Application in aircraft control
Domain specific knowledge	Not fully exploited	Exploit prior knowledge about the effect of different controls, relationships between states ...
System integration	Tested each module separately	Full test of the complete control system under different flight conditions
Robustness	Not tested	Test performance with noise. Explore methods of neural network training that improve robustness
On-line design of optimal controllers	Hopfield optimal trajectory generation	Pole placement LQG/LTR H_∞
Bi-directional associative memories (BAM)	Not tested	Use BAM to implement inverse trim
Dimensionality of inverse model	Full dimensionality	Reduce dimensionality to improve learning and control
On-line tests	Only off-line training	Test on-line training and control
Optimization of objective function parameters	Not explored	• Explore fuzzy logic • Multi-objective optimization
Neuro Control Toolbox	Prototypes using different languages: C, Fortran, Matlab™	CASYS with C/C++

6. REFERENCES

Ahmed-Zaid, F., Ioannou, P.A., Polycarpou, M.M., et al. 1992. "Identification and Control of Aircraft Dynamics Using Radial Basis Functions Neural Networks." *AIAA Guidance, Navigation, and Control Conference.*

Atkeson, C.G. 1989. "Using Local Models to Control Movement." *Neural Information Processing.* D.S. Touretzky, ed. San Mateo, CA: Morgan Kauffman Pub.

Atkeson, C.G. 1991. "Memory-Based Learning Control." *Proceedings of the 1991 American Control Conference.* Boston, MA.

Barhen, J., Gulati, S. and Zak, M. 1989. "Neural Learning of Constrained Non-Linear Transformations." *Computer*, Vol. 22, No. 6: pp. 67-76.

Barron, R., Celluci, R.L. and Jordan, P.R. 1990. "Applications of Neural Networks to FDIE and Flight Control." *NAECON 90.* Cayton, OH.

Bertsekas, D. 1982. *Constrained Optimization and Lagrange Multiplier Methods.* New York: Academic Press.

Botros, S.M. and Atkeson, C.G. 1991. "Generalization Properties of Radial Basis Functions." *In Advances in Neural Information Processing Systems 3.* , .P. Lippman, J.E. Moody and D.S. Touretzky, ed. San Mateo, CA: Morgan Kaufmann Pub.

Broomhead, D.S. and Lowe, D. 1988. "Multivariable Functional Interpolation and Adaptive Newtworks." *Complex Systems*, Vol. 2: pp. 321-355.

Caglayan, A.K. and Allen, S.M. 1990. "A Neural Net Approach to Space Vehicle Guidance." *American Control Conference.* San Diego, CA.

Calise, A.J., Kim, B.S., Kam, M., et al. 1992. "Neural Networks for Feedback Linearization in Aircraft Control." *AIAA Guidance, Navigation, and Control Conference.*

Caprile, B. and Girosi, F. *A Nondeterministic Minimization Algorithm.* Artificial Intelligence Memo 1254, MIT: 1990.

Chandler, P.R., Pachter, M. and Mears, M. 1993. "A Hopfield Neural Network for Adaptive Control." *AIAA Guidance, Navigation and Control Conference.* Monteray, CA.

Chen, F.C. and Khalil, H.K. 1992. "Adaptive Control of Nonlinear Systems Using Neural Networks." *International Journal of Control*, Vol. 55, No. 6: pp. 1299-1317.

Chu, S.R. and Shoureshi, R. 1992. "Applications of Neural Networks in Learning of Dynamical Systems." *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 1: pp. 160-164.

Chu, S.R., Shoureshi, R. and Healey, A.J. 1992. "A Hopfield-Based Neuro-Diagnostic System." *American Control Conference.*

Chu, S.R., Shoureshi, R. and Tenorio, M. 1990. "Neural Networks for System Identification." *IEEE Control Systems Magazine*, Vol. April: pp. 31-35.

Cichocki, A. and Unbehauen, R. 1992. "Neural Networks for Computing Eigenvalues and Eigenvectors." *Biological Cybernetics*, Vol. 68: pp. 155-164.

DiGirolamo, R. 1992. "Flight Control Law Synthesis Using Neural Network Theory." *AIAA Guidance, Navigation & Control Conference.*

Girosi, F. 1992. "Some Extensions of Radial Basis Functions and Their Applications in Artificial Intelligence." *Computers and Mathematics with Applications*, Vol. 24, No. 12: pp. 61-80.

Girosi, F. and Poggio, T. 1989. "Representation Properties of Networks: Kolmogorov's Theorm is Irrelevant." *Neural Computation*, Vol. 1: pp. 465-469.

Girosi, F. and Poggio, T. 1990. "Networks and the Best Approximation Property." *Biological Cybernetics*, Vol. 63: pp. 169-176.

Girosi, F., Poggio, T. and Caprile, B. *Extensions of a Theory of Networks for Approximation and Learning: Outliers and Negative Examples*. MIT AI Memo 1220: 1990.

Hertz, J., Krough, A. and Palmer, J.P. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA. Lecture Notes: Santa Fe Institute Studies in the Sciences of Complexity: 1991. Volume 1:

Hoskins, D.A., Hwang, J.N. and Vagners, J. 1992. "Interactive Inversion of Neural Networks and its Applications to Adaptive Control." *IEEE Transactions on Neural Networks*, Vol. 3, No. 2: pp. 292-301.

Hunt, K.J., Sbarbaro, D., Zbikowski, R., et al. 1992. "Neural Networks for Control Systems - A Survery." *Automatia*, Vol. 28, No. 6: pp. 1083-1112.

Hutchinson, M., Kalma, J. and Johnson, M. 1984. "Monthly Estimates of Windspeed and Wind Run for Australia." *Journal of Climatology*, Vol. 4: pp. 311-324.

Jacobs, R.A. and Jordan, M.I. 1993. "Learning Piecewise Control Strategies in a Modular Neural Network Architecture." *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 23: pp. 337-345.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J., et al. 1991. "Adaptive Mixtures of Local Experts." *Neural Computation*, Vol. 3: pp. 79-87.

Jordan, M.I. 1989. *Indeterminate Motor Skill Learning Problems*. Hillsdale, NJ: Attention and Performance XIII, Lawrence Erlbaum.

Jordan, M.I. and Rumelhart, D.E. 1992. "Forward Models: Supervised Learning with a Distal Teacher." *Cognitive Science*, Vol. 16, No. July/Sept: pp. 307-354.

Kawato, M. and Gomi, H. 1992. "A Computational Model of Four Regions of the Cerebellum Based on Feedback Error Learning." *Biological Cybernetics*, Vol. 68: pp. 95-103.

Kirk, D.E. 1970. *Optimal Control Theory: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall.

Kuschewski, J.G., Hui, S. and Zak, S.H. 1993. "Application of Feedforward Neural Networks to Dynamical System Identification and Control." *IEEE Transactions on Control Systems Technology*, Vol. 1, No. 1: pp. 37-49.

Lan, M. and Chand, S. 1990. "Solving Linear Quadratic Discrete-Time Optimal Controls using Neural Networks." *Proceedings of the 29th Conference of Decision and Control*. Honolulu, HI. December.

Levin, A.U. and Narendra, K.S. 1993. "Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization." *IEEE Transactions on Neural Networks*, Vol. 4, No. 2: pp.

Li, K.-C. 1992. "On Principal Hessian Directions for Data Visualization and Dimension Reduction: Another Application of Stein's Lemma." *Journal of the Americal Statistical Association*, Vol. 87: pp. 1025-1039.

Linse, D. 1990. *Neural Networks for Flight Control*. NASA. CP 3063.

Marzuki, K. and Omatu, S. 1992. "A Neural Network Controller for a Temperature Control System." *IEEE Control Systems Magazine*, Vol. 6: pp. 58-64.

Mears, M.J., Smith, R., Chandler, P.R., et al. 1993. "Hopfield Neural Network for Adaptive Control." *AIAA Guidanc Navigation and Control Conference*. Monterey, CA.

Mel, B.W. and Omohundro, S.M. 1991. "How Receptive Field Parameters Affect Neural Learning." *In Advances in Neural Information Processing Systems 3*. R.P. Lippman, J.E. Moody and D.S. Touretzky, ed. San Mateo, CA: Morgan Kaufmann Pub.

Micchelli, C.A. 1986. "Interpolation of Scattered Data: Distance Matrices and Conditionally Positive Definite Functions." *Constructive Approximation*, Vol. 2: pp. 11-22.

Moody, J. and Darken, C. 1989. "Fast Learning in Networks of Locally-Tuned Processing Units." *Neural Computation*, Vol. 1: pp. 281-294.

Moonen, M., DeMoor, B., Vandenberghe, L., et al. 1992. "On-and-Off-Line Idenfication of Linear State-Space Models." *International Journal of Control*, Vol. 49: pp. 219-232.

Narendra, K.S. and Mukhopadhyay, S. 1992. "Intelligent Control Using Neural Networks." *IEEE Control Systems*, Vol. 4: 11-18.

Pao, Y.-H., Phillips, S.M. and Sobajic, D.J. 1992. "Neural-Net Computing and the Intelligent Control of Systems." *Int. J. Control*, Vol. 56, No. 2: 263-289.

Platt, J. 1991. "A Resource-Allocating Network for Function Interpolation." *Neural Computation*, Vol. 3: pp. 213-225.

Powell, M.J.D. 1987. "Radial Basis Functions for Multivariable Interpolation." *Algorithms for Approximation*. J.C. Mason and M.G. Cox, ed. Oxford: Clarendon Press. pp.143-167.

Psaltis, D., Sideris, A. and Yamamura, A.A. 1988. "A Multilayered Neural Network Controller." *IEEE Control Systems Magazine*, Vol. 4: pp. 17-21.

Rokhsaz, K. and Steck, J.E. 1993. "Application of Artificial Neural Networks in Non-Linear Aerodynamics and Aircraft Design." *Aerotech '93*. Costa Mesa, CA.

Sadeghi, T., Tascillo, M., Simons, A., et al. 1992. *Adaptive Reconfigurable Flight Control for High Angle of Attack Aircraft Agility*. Nato AGARD Report 789 (Stability in Aerospace Systems).

Saha, A., Christian, J., Tang, D.S., et al. 1991. *In Advances in Neural Information Processing Systems 3*. San Mateo, CA: Morgan Kaufmann Pub.

Samad, T. and Mathur, A. 1991. "Parameter Estimation for Process Control with Neural Networks." *SPIE, Applications of Neural Networks II*, Vol. 1469: pp. 766-777.

Samarov, A.M. *Exploring Regression Structure Using Nonparametric Functional Estimation*. TR-69, Sloan School of Management, MIT: 1991.

Sanner, R.M. and Slotine, J.J. 1992. "Gaussian Networks for Direct Adaptive Control." *IEEE Transactions on Neural Networks*, Vol. 3, No. 6: pp. 837-863.

Schagen, I.P. 1980. "Stochastic Interpolating Functions: Applications in Optimization." *Journal of Institute of Mathematics and its Applications*, Vol. 3.

Schiffman, W.H. and Geffers, H.W. 1993. "Adaptive Control of Dynamic Systems by Backpropagation Networks." *Neural Networks*, Vol. 6: pp. 517-524.

Steinberg, M. 1992. "Potential Role of Neural Networks and Fuzzy Logic in Flight Control Design and Development." *1992 Aerospace Design Conference*. Irvine, CA.

Swindelhurst, A., Roy, R., Ottersten, B., et al. 1992. "System Identification via Weighted Subspace Fitting." *American Control Conference*.

Troudet, T., Garg, S. and Merrill, W. 1993. "Neurocontrol Design and Analysis for a Multivariable aircraft control problem." *Journal of Guidance, Control and Dynamics*, Vol. 16, No. 4: pp. 738-747.

Wang, L.X. and Mendel, J.M. 1991. "Cumulant-Based Parameter Estimation Using Structured Networks." *IEEE Transactions on Neural Networks*, Vol. 2, No. 1: pp. 73-83.

Zhao, Y. 1992. *On Projection Pursuit Learning*. Ph.D Thesis. Massachusetts Institute of Technology, Department of Mathematics,

Appendix A: Radial Basis Functions Networks for Function Approximation

The Radial Basis Functions (RBF) approach to approximating continuous functions belongs to the class of non-parameteric approximation techniques. Non-parametric techniques also include feedforward neural networks, projection pursuit regression, nearest neighbor approximation and local weighted regression. RBF approximation consists of modeling an input output mapping as a linear combination of radially symmetric functions (Powell 1987; Girosi and Poggio, 1990; Broomhead and Lowe, 1988; Moody and Darken, 1989). It was first developed as an exact interpolation approach; that is, it reproduces the outputs of the given examples exactly. The output of the interpolating function is described by the following equation :

$$y(x) = \sum_{i=1}^{N} C_i \phi(\|x - x_i\|) \qquad \text{(A-1)}$$

where $x \in \Re^n$ and $y \in \Re$ represent the input and output respectively, $\|\cdot\|$ represent the Euclidean norm, C_i's are the coefficients to be estimated, and N is the number of data points in the training set.

Depending on the type of RBF used, a polynomial term of the form

$$\sum_{j=1}^{p} \mu_j P_j^m(x_i) \qquad \text{(A-2)}$$

is added to equation (A-1), where the μ_j are unknown coefficients, and P_j^m are polynomials of degree $\leq m$. In such a case, since the number of parameters is larger than the number of data points, the following extra constraints are added to make the parameter estimation problem well posed (Powell, 1987).

$$\sum_{j=1}^{n} C_j P_i^m(x_j) = 0 \qquad i = 1, \cdots p \qquad \text{(A-3)}$$

Examples of RBFs include :

• Gaussians	$\Phi(r_{ij}) = \exp\left(\frac{-r_{ij}^2}{c^2}\right)$
• Hardy Multiquadrics	$\Phi(r_{ij}) = \sqrt{r_{ij}^2 + c^2}$
• Hardy Inverse Multiquadrics	$\Phi(r_{ij}) = \frac{1}{\sqrt{r_{ij}^2 + c^2}}$
• Cubic Splines	$\Phi(r_{ij}) = r_{ij}^3$
• Linear Splines	$\Phi(r_{ij}) = r_{ij}$
• Thin Plate Splines	$\Phi(r_{ij}) = \begin{cases} r_{ij}^{2k-d} \log(r) & r \text{ even} \\ r_{ij}^{2k-d} & r \text{ odd} \end{cases}$

$2k > d$, d is the dimension of the input and k is a smoothness parameter.

where $r_{ij} = \| x_i - x_j \|$.

Some of these RBFs (e.g. Gaussians and multiquadrics) have an explicit width parameter c that needs to be determined. However we can also fix this width parameter, for example to have a value of one and scale the data instead. In the applications we report in this work, we use Gaussian RBFs.

To find the coefficients C_j for exact interpolation, we have to invert a square matrix which is theoretically guaranteed to be nonsingular for a wide class of radial basis functions, given, of course distinct data (Micchelli, 1986). Since the equations are linear, there are a number of batch and recursive algorithms that exist for finding the exact value of the coefficients. The linearity of the function with respect to the coefficients C_j guarantees the convergence to the globally optimal parameters. Since many of the optimization algorithms require the inversion of a matrix of rank n , the computational complexity for finding the optimal parameters is $O(n^3)$, where n is the number of data points and also the number of coefficients in the case of exact interpolation. Exact interpolation may not be desirable if the data are noisy or if the computational burden is high.

A.1 Mathematical Interpretation of RBFs

As mentioned by Poggio and Girosi, (1989), many of the radial basis functions are the Green functions obtained by solving different regularization problems of the form:

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \| Pf \|^2 \qquad \text{(A.1-1)}$$

Where P in the above equation is a radially symmetric differential operator. For example Gaussian RBFs result from operators P of the form:

$$\int_{R^d} dx \sum_{m=1}^{\infty} a_m \left(P^m f(x)\right)^2 \qquad \text{(A.1-2)}$$

where $P^{2m} = \nabla^{2m}$ and $P^{2m+1} = \nabla \nabla^{2m}$, ∇^2 is the Laplacian operator and the coefficients $a_m = \frac{\sigma^{2m}}{m!\, 2^m}$. Similar regularization functions may also be derived for other types of RBFs. Since regularization is also related to Bayesian estimation, we can think of RBFs as a special case of Bayesian estimation (Girosi, Poggio and Caprile, 1990), where the prior probability of the function f(x) is assumed to be

$$P(f) \alpha \exp(-\lambda \| Pf \|^2 \qquad \text{(A.1-3)}$$

Another interesting interpretation for some forms of RBFs made by Schagen (1980) is to regard the given training examples as point realizations of a stationary stochastic process Z(x). The stationarity of Z(x) implies that the mean and variance of the process at any point are constants and that the covariance between two points is only a function of the difference between these two points. If we make the stronger assumption that the covariance between two points $g(x_i,x_j)$ depends only on the distance between these two points(ie., $g(x_i,x_j) = g(\|x_i - x_j\|)$) and given that the function g(r) satisfies the covariance properties, namely $g(0) = 1$, $g(r) \leq 1$ for $r \geq 0$ and the covariance matrix is nonnegative definite, some RBF solutions may be interpreted as the best linear unbiased estimate of the stochastic process given the data points. For the derivation of this result, we refer the reader to the paper by Schagen (Schagen, 1980). Gaussian RBFs satisfy all these assumptions and constraints. Note that not all RBF functions mentioned above satisfy all the assumptions required for this interpretation. RBFs with increasing function values as the distance from the center increases cannot model a covariance since $g(r) \geq g(0)$ for some $r > 0$.

Both of the above interpretations of radial basis functions make some a priori assumptions about the degree of smoothness of the function to be approximated. These a priori assumptions determine the shape of the radial basis function used.

A.2 Extensions to RBF: GRBF and HyperBF

To reduce the problems of exact interpolation, many researchers have suggested using a smaller number of basis functions (Broomhead and Lowe, 1988; Girosi and Poggio, 1989; Moody and Darken, 1989). In this case it is not possible to reproduce the exact outputs in general. To choose the centers of the basis functions we can use optimization techniques (Girosi and Poggio, 1989) or also heuristic algorithms based on the distribution of the data (Moody and Darken, 1989).

The RBF approximation with movable centers has been called Generalized Radial Basis Functions (GRBF) (Girosi and Poggio, 1989). The estimation of the location of the centers of the GRBFs using least square error optimization techniques is not an easy problem. This is because the error surface is not convex, and the number of parameters to be estimated is large. From our experience with different computer simulations and using second order nonlinear optimization techniques, we found that it is very hard to adjust the centers and that usually the gain in performance is small.

Another extension to the RBF approach, described also by Poggio and Girosi (Girosi and Poggio, 1989) is known as Hyper Basis Functions (HyperBF). This is a further generalization of the GRBF technique, which includes using radial basis functions of different widths or also non-radial basis functions. Similar types of basis functions have been described by Saha, Christian, Tang, et al. (1991) for image coding and analysis and have been termed Oriented Non-Radial Basis Functions (ONRBF). Saha, et al. (1991) suggest a gradient descent algorithm for finding the parameters of the ONRBFs. Varying the widths of the RBFs is also equivalent to using a general norm rather than the Euclidean norm to compute the distance of a point from the center of the basis function.

The equation describing the output in terms of the basis functions and the different inputs is as follows :

$$y(x) = \sum_{i=1}^{N} C_i \phi(\|x - x_i\|_W) \qquad \text{(A.2-1)}$$

where

$\|x - x_i\|_W^2 = (x_i - x_j)^T W^T W (x_i - x_j)$ and W is a square matrix.

From our practical experience, it is found that the W matrix plays a very important role in the quality of generalization. This is especially true for functions which do not meet the smoothness assumptions implied a priori by using a certain type of RBFs.

We use HyperBF networks with Gaussian units and diagonal weight matrix W in modeling the aircraft longitudinal inverse dynamics, the inverse trim function, and the associative parameter ID module. In the next section we describe different methods for estimating the diagonal weight matrix W, given the input-output data.

A.3 Estimating a Diagonal Weight Matrix W for Gaussian RBFs

There are many possible ways for estimating the weight matrix W. One class of methods, based on optimization techniques, is to find a W which minimizes the sum of the square of the errors between the output of the RBFs and the training set output. Nonlinear optimization

techniques, that might be used include gradient descent, second-order nonlinear optimization techniques or variations of random search. Gradient and second order methods are not guaranteed to converge to the global minima and are sensitive to the initial choice of parameters. Also, for large amounts of data and RBFs, the amount of computation involved in second order methods becomes very large. In random search, the amount of computation for each step is relatively small, but a very large number of steps may be needed to converge, especially when the number of parameters to be estimated is large. The main advantage of random search is that it can escape from local minima. Caprile and Girosi (1990) present a simple random search technique that has been found to work well in practice.

Another alternative method for determining the best diagonal weight matrix W is to use cross-validation techniques to estimate the w_i's in the different input dimensions. This has the advantage over minimum training error techniques in that it attempts to minimize the predicted mean square error, as opposed to the mean square error of the training set only, and therefore may generalize better. Hutchinson, Kalma and Johnson (1984) have attempted to use generalized cross validation to find scaling parameters for one input variable. More work is needed to generalize this technique to more than one input variable in an efficient way. However, in general, cross validation techniques tend to be computationally very expensive, and it may be very difficult to adapt these techniques to real time applications.

Many researchers have explored some heuristic methods for the estimation of the RBF widths and center locations. Moody and Darken (1989) describe methods based on adaptive clustering of the input data. In their analysis, they totally ignore the characteristics of the function to be approximated. Platt (1991) describes a resource allocating network that adaptively adds basis functions based on a novelty measure. The novelty measure is based on two factors: the accuracy of the approximation and the distance of the new experience from the previous data points. The width of the RBFs is proportional to the distance to the k-nearest neighbor. Although the output data in this method are used in the choice of the center locations, the estimation of the RBF widths is still completely dependent on the input distribution only. Hutchinson (1993) proposes a heuristic algorithm for finding a reasonable set of initial values of the parameters of the RBFs. His algorithm is a generalization of Moody and Darken algorithm and allows for the possibility of estimating the widths of RBFs based on the output as well as the input data. Methods that depend only on the input distribution to determine the RBF width parameters may not work well if the dependence of the function to be approximated in the different directions of the input space is not uniform. Mel and Omohundro (1991) describe a method that depends on the second order derivatives of the function to be approximated with respect to the different input variables.

In the simulations reported in this work, we use a heuristic technique for estimating for estimating a diagonal W for Gaussian RBFs. This method is based on approximating the first order partial derivatives of the function to be approximated with respect to the different input variables. Although this method is not proven to optimize any cost function, it is found to approximate and sometimes surpass the results obtained using nonlinear optimization techniques (Botros and Atkeson, 1991). The diagonal width parameters are assumed to depend on the average variation of the function in each direction, as measured by the sum of the square of the first partial derivatives in each direction, in addition to the variance of the input data in the different dimensions. Define the average gradient g to be:

$$\mathbf{g} = \int_{R^d} \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n} \right]^T dx \qquad \text{(A.3-1)}$$

and the normalized vector $\bar{\mathbf{g}} \equiv \frac{\mathbf{g}}{\|\mathbf{g}\|}$. We find empirically that a good approximation for the diagonal of W is as follows :

$$w_{ii} = \bar{g}_{ii} \frac{k}{\sqrt{E\{(x_i - t_i)^2\}}} \qquad \text{(A.3-2)}$$

where the subscript i denotes the i^{th} input variable, $E\{ \cdot \}$ represents the expected value, and t_i is the i^{th} component of the centers of the RBFs. The parameter k is a constant that can be determined by cross validation or least mean square optimization. From simulations we find that the approximation is not very sensitive to a range of the values of k. However, the choice of a good k is important for the conditioning of the coefficients of the RBF. The bigger the value of k, the smaller will be the equivalent width of the RBFs, and the estimation of the coefficients will be less singular.

To understand why this suggested form of W makes sense, we can divide the equation for W into two terms. The first term is the normalized gradient functional and the second is a normalization factor that normalizes the input space so that the different inputs have approximately similar range of values. The gradient functional term results in making the first order terms in the regularizer operator for Gaussian RBFs (equation b-5) have approximately equal magnitude. This, in turn, satisfies the assumptions made by the regularizer. Intuitively, the width of the Gaussian functions will be smaller in the directions where the approximated function varies the most. Note also that if one variable is irrelevant, its derivative function will be zero, and therefore the corresponding w component will also be zero.

The idea of separating the estimation of the norm metric from the estimation of the other function approximation parameters has been recognized by many other researchers Girosi (1992) Moody and Darken (1989); Samarov (1991) Li (1992) Zhao (1992). Some of these researchers have also suggested the use of different forms of derivative functionals for other function approximation techniques (Samarov, 1991; Zhao,1992; Li, 1992). Both Samarov and Li have described methods for estimating these derivative functionals or expected derivatives from the data and mentioned the assumptions under which these estimations are valid. We use an iterative technique for the estimation of the matrix W. This technique starts by first estimating the function to be approximated using a diagonal W matrix equal to the inverse of the variance of the input data in the different input directions, and then estimating the derivative functional using the approximated function. We then use the derivative functional to update the value of the W matrix and use this latter to improve the function approximation. In practice, this procedure usually converges in 3 or 4 iterations. However a more detailed mathematical analysis is needed to understand the convergence properties of this algorithm.